UNIX: introduzione elementare

Guida introduttiva al sistema operativo Unix per principianti

Marco Liverani

Seconda edizione – Settembre 2005

Questo fascicolo è stato prodotto utilizzando il software LaTeX2$_\varepsilon$.

Prima edizione Maggio 1995
Seconda edizione Settembre 2005 (17/9/2005 – revisione del 24/6/2012)

Indice

Introduzione

Unix non è certo un'invenzione recente. Le prime versioni di questo sistema operativo furono sviluppate negli Stati Uniti intorno ai primi anni '70 presso gli ormai mitici AT&T Bell Laboratories. Sono passati quindi più di trent'anni da allora, un secolo se rapportato ai tempi rapidissimi di evoluzione dell'informatica. Nel frattempo sono cambiate le architetture hardware dei computer e persino gli scopi per cui questi vengono utilizzati, in questi trent'anni è cambiato quasi tutto ciò che riguarda la microinformatica. Eppure lui, Unix, è ancora lì, perfettamente integrato con i nuovi sistemi ed anzi, da una decina di anni a questa parte sta vivendo una seconda giovinezza (o una seconda maturità, sarebbe il caso di dire), non più soltanto per il peso che negli anni '90 aveva assunto il settore delle *workstation* basate su architettura RISC, ma piuttosto per lo sviluppo che hanno avuto nuove generazioni di sistemi operativi "Unix-like" per macchine con l'architettura di un personal computer. Sto pensando naturalmente al sistema operativo Linux, nelle sue innumerevoli versioni e "distribuzioni", ma anche ai vari sistemi operativi *open source* di derivazione BSD, come Open BSD, FreeBSD, NetBSD ed il più recente Darwin, con la sua controparte *open source* denominata Open Darwin, realizzato dalla Apple come nucleo su cui basare il suo meraviglioso sistema operativo Mac OS X. Ma forse i motivi di questa longevità (più unica che rara, lo ripeto, nel mondo dell'informatica dove tutto invecchia rapidamente) li dobbiamo ricercare anche nel portentoso sviluppo che le reti (locali e geografiche) hanno avuto nell'ultimo decennio; e Unix, in un ambiente di rete, la fa da padrone. La maggior parte dei server su Internet sono macchine Unix, su Unix sono basati la maggioranza dei server di posta elettronica, dei server web e dei *repository* per la condivisione e lo scambio di file; in una macchina in ambiente Unix tutti i servizi di rete sono perfettamente integrati col sistema (davvero la rete diventa un'estensione naturale della *workstation*) e questo spiega un vecchio ma efficace slogan di Sun Microsystems, leader mondiale nella produzione di sistemi Unix: "*The network is the computer*".

Proprio perché è stato concepito in modo esemplare, Unix è anche un ottimo "oggetto didattico", un sistema da usare per imparare veramente cosa è un computer, cosa è un sistema operativo; studiare tutto questo su un PC in ambiente Windows può essere forse più comodo, ma ci darebbe un punto di vista limitato, già molto indirizzato verso un certo tipo di applicazioni, di sicuro non completo. Dopo aver studiato ed operato su una macchina Unix, potremo capire cosa è un PC e anche come funziona il sistema operativo Windows con maggiore facilità, potremo apprezzarne meglio le differenze, la maggiore facilità d'uso ottenuta al costo di una potenza operativa inferiore (anche a parità di risorse hardware). È possibile così accorgersi quanto Unix abbia influenzato anche lo sviluppo del sistema operativo di Microsoft, soprattutto nelle versioni più recenti, che hanno integrato in modo molto efficace i concetti di multiutenza, di servizio e di memoria protetta, che caratterizzano i sistemi operativi Unix da moltissimi anni.

Per capire a fondo il perché di certi aspetti di Unix è forse utile conoscerne gli utenti. Non conosco un altro sistema operativo che sia stato così pesantemente modellato dai suoi utenti, per certi aspetti si potrebbe dire che Unix è parzialmen-

te "fatto in casa", ogni installazione di Unix è ritagliata, modificata ed integrata dai suoi utenti, mediante un insieme di applicazioni piccole o grandi, programmate direttamente o reperite per i canali del software *open source* attraverso la rete Internet. E infatti gli utenti Unix, quelli che il sistema operativo hanno imparato ad apprezzarlo nei suoi aspetti più essenziali, sono un po' particolari, formano quasi una allegra "setta", si sentono (sono) la crema degli utenti di computer, amano essere autodidatti, adorano integrare il sistema con programmi sviluppati in proprio, scrivono correntemente in C e in Perl, il loro editor preferito è *vi* (e presto capirete quali implicazioni ha un'affermazione di questo tipo!). Parlano con un codice linguistico molto particolare, spesso si capiscono solo tra di loro. Soprattutto, un vero utente Unix troverebbe aberrante questa guida: è troppo breve ed incompleta, e per questo non rende giustizia alla grande potenza del sistema.

Questa brevissima guida non è un manuale esaustivo su Unix, ed in effetti non è proprio un manuale su Unix. È una guida introduttiva. Su ogni argomento toccato dai capitoli di questo fascicolo sono stati scritti libri interi ed è a questi testi (qualcuno lo riporto tra le note bibliografiche) che vi consiglio di riferirvi per approfondire la conoscenza di Unix. Utilizzate questo fascicolo solo per avere una prima informazione sintetica su cosa vi si apre davanti accedendo ad una macchina Unix. Le informazioni riportate nella guida non sono errate (almeno questo è ciò che mi auguro!), ma sono profondamente incomplete. I comandi della *shell* Unix sono caratterizzati da una serie infinita di opzioni che ne modificano l'esecuzione e che li rendono veramente flessibili e potenti; ho riportato in queste pagine solo pochi comandi e di ognuno di questi ho elencato solo le opzioni principali. Sta a voi, alla vostra curiosità, alle vostre necessità, il compito di scoprire di volta in volta nuove varianti di uno stesso comando. In altre parole si può dire che in questa guida non viene spiegato in dettaglio *come* compiere determinate operazioni, ma piuttosto si è preferito puntare l'attenzione su *quali* operazioni è possibile compiere, rimandando alla documentazione del sistema per tutto ciò che riguarda i dettagli tecnici e sintattici. Consiglio di leggere questo fascicolo per intero e comunque nell'ordine in cui sono riportati i capitoli: non essendo un manuale di consultazione, ma una guida introduttiva, non avrebbe nessun senso cominciare direttamente da un capitolo diverso dal primo.

Nella prima edizione di questa guida (scritta più di dieci anni fa!), per ragioni di sintesi e di chiarezza spesso nella descrizione dei comandi e delle funzionalità avevo fatto dei paralleli con i comandi e le funzionalità del sistema operativo MS-DOS. Oggi quel sistema operativo non esiste più e dunque nella presente edizione ho rivisto questo approccio dal momento che i neofiti di oggi (è a loro che si rivolge questa guida), se anche conoscono il sistema operativo Windows, è molto probabile che non conoscano neanche uno dei comandi della *shell* (il cosiddetto "*prompt* dei comandi", nel gergo Windows) e quindi il parallelo non sarebbe di nessuna utilità.

Prima di concludere questa breve introduzione voglio raccomandarvi di sperimentare ogni cosa sulla vostra macchina, senza alcun timore di danneggiare i dati degli altri utenti del sistema. Unix dispone di numerosi meccanismi di sicurezza e quindi non vi sarà facile compromettere il buon funzionamento del sistema o danneggiare il lavoro dei vostri colleghi.

Ho cercato di porre la massima cura ed attenzione nella realizzazione di questo fascicolo. Nondimeno saranno sicuramente presenti numerosi refusi tipografici ed inesattezze sostanziali nel testo. Invito fin d'ora, chiunque riscontrasse errori di qualsiasi tipo, a segnalarmeli con un messaggio di posta elettronica agli indirizzi `marco@aquilante.net` e `liverani@mat.uniroma3.it`.

Convenzioni tipografiche

Nel comporre il testo di questo fascicolo sono state impiegate alcune convenzioni tipografiche. I caratteri `typewriter` sono stati utilizzati per indicare i comandi del sistema (ciò che si deve digitare). Negli esempi invece lo stesso carattere è stato utilizzato per rappresentare i messaggi visualizzati dal sistema; ciò che negli esempi si intende digitato dall'utente è stato scritto con il carattere *`typewriter corsivo`*. I singoli tasti che devono essere battuti per impostare determinati comandi sono rappresentati con una cornice: [Return]. Ho riportato delle note a margine delle pagine per consentire una identificazione più immediata degli argomenti trattati.

Le numerose figure che compaiono alla fine dei capitoli, rappresentano alcune *workstation* Unix ormai fuori produzione. Le immagini sono state selezionate sulla rete Internet nei siti web delle rispettive case costruttrici, o in gallerie di foto "d'epoca" pubblicate da qualche appassionato. Spero con questo di non aver violato alcun vincolo imposto dal copyright.

Premessa alla seconda edizione

Ho provato più volte a rimettere mano alla prima edizione di questa guida per integrarla, ampliarla e soprattutto per correggere gli errori che qualche lettore attento mi ha segnalato. Non si tratta di un compito facile, dal momento che ciò che avevo scritto dieci anni fa era un documento per certi versi un po' ingenuo, ma anche molto equilibrato nel bilanciare le informazioni presenti nei cinque capitoli di cui era costituito. La pigrizia e la paura di rompere questo equilibrio che molti avevano apprezzato mi ha indotto più volte a rimandare questa attività di revisione. Oggi finalmente ho trovato la strada per rielaborare ciò che avevo scritto e per produrre qualcosa che spero sia migliore.

Ho corretto tutti gli errori che mi sono stati segnalati ed ho aggiunto o solo ampliato alcune parti della guida. Sono molto grato a quanti mi hanno incoraggiato a diffondere questo lavoro che ha avuto un successo enorme e insperato. Alcuni miei ex-studenti mi hanno segnalato che questa guida è stata utilizzata in alcune occasioni perfino da grandi aziende di informatica e telecomunicazioni come supporto didattico nei corsi di formazione per i nuovi assunti. D'altra parte sono numerosi i siti web che hanno "linkato" l'indirizzo della pagina Internet su cui ho pubblicato questa guida, ed altrettanti quelli che hanno deciso di copiarla integralmente per proporla con un altro stile tipografico ai propri utenti e lettori. Tutto questo ovviamente mi fa molto piacere e dunque invito quanti decideranno di utilizzare la guida come supporto per corsi di istruzione e di formazione[1] a segnalarmelo con un messaggio di posta elettronica.

M.L.

Roma, Settembre 2005

[1] La guida è gratuita e può essere liberamente riprodotta e distribuita purché questo venga fatto senza fini di lucro e rispettando il copyright dell'autore e la paternità dell'opera.

Capitolo 1

Organizzazione del sistema

In questo primo capitolo voglio cercare di dare una brevissima infarinatura generale sulla struttura teorica di un sistema Unix, volendo esagerare potrei anche dire sulla "filosofia di un sistema Unix". Una macchina che opera con questo sistema operativo è infatti concepita in modo del tutto diverso da un computer per uso personale; non voglio essere frainteso: oggi esistono numerose versioni del sistema operativo Unix anche per PC o per i computer Apple Macintosh, ma è difficile parlare di *personal computing* in presenza di Unix, visto che ben poco spazio lascia a ciò che siamo soliti fare con il nostro PC. Soprattutto è diverso il modo di usare il sistema da parte dell'utente.

Di sicuro Unix è molto più potente, più robusto e più efficiente di Windows, è anche più versatile e forse anche più divertente, ma di conseguenza è più difficile da apprendere; tuttavia lavorare con questo potente strumento può dare enormi soddisfazioni: una volta che avrete imparato ad operare su un sistema Unix sarà ben difficile accettare le anguste ristrettezze a cui ci costringono altri sistemi operativi.

1.1 Introduzione al sistema operativo Unix

In termini assai riduttivi possiamo dire che un sistema operativo è il software che, caricato in memoria ed eseguito al momento dell'accensione del calcolatore (*bootstrap*), permette ai programmi applicativi di girare e di sfruttare le risorse hardware della macchina. Il sistema operativo in un certo senso impone anche all'utente il modo di operare sulla macchina, stabilendo ciò che l'utente può fare e ciò che invece non può fare, stabilendo l'ordine con cui devono essere eseguite certe operazioni e così via.

Di questi aspetti ci si accorge poco utilizzando su un personal computer con vecchi sistemi operativi mono-utente come MS-DOS, le prime versioni di Windows e le vecchie versioni del System della Apple: in questi ambienti infatti poche cose è consentito fare, ma l'utente le può eseguire tutte, senza alcuna restrizione.

D'altra parte tali sistemi operativi sono stati appositamente progettati per girare su un personal computer, su una macchina che quindi sarebbe stata utilizzata da un unico utente per volta e spesso da un solo utente in assoluto. MS-DOS e le prime versioni di Microsoft Windows, ad esempio, erano anche concepiti per poter eseguire un unico programma (*task*) per volta[1]: mentre si utilizzava un editor, non era possibile *contemporaneamente* utilizzare un programma di comunicazione o eseguire un programma di utilità. Oggi queste limitazioni sono superate praticamente da tutti i sistemi operativi in circolazione, ma le caratteristiche di multiutenza e di

[1] Le prime versioni di Windows non erano un vero e proprio sistema operativo, ma una interfaccia utente grafica che si appoggiava sul sistema operativo mono-utente e mono-tasking MS-DOS.

multitasking su macchine accessibili ad una vasta fascia di utenti (i personal computer, ad esempio) per molti anni sono state dominio esclusivo dei sistemi operativi Unix.

Unix nasce intorno ai primi anni '70 contemporaneamente all'introduzione dei *mini computer*, nuove macchine più compatte ed agili dei grandi *mainframe* prodotti fino ad allora. Le prerogative di Unix che ne determinarono subito un grande successo sono sostanzialmente quelle di essere un sistema operativo compatto e modulare, facilmente adattabile alle risorse hardware ed alle esigenze operative dell'ambiente in cui viene installato. Con l'introduzione di Unix e dei mini computer si diffonde l'uso dei terminali alfanumerici che consentono una comunicazione più facile ed efficiente tra l'utente ed il sistema (fino ad allora si erano usate quasi esclusivamente le schede perforate). Si fa largo anche il concetto di *sistema aperto* e di *elaborazione distribuita*: Unix è predisposto per comunicare facilmente con altri sistemi e ad essere interconnesso con altre macchine per operare su risorse non più centralizzate su un unico grosso sistema, ma distribuite su più macchine di media potenza; la configurazione del sistema e i formati utilizzati per i suoi file sono aspetti noti e documentati (in gergo si dice "aperti") e dunque chiunque viene messo nelle condizioni di poter intervenire su di essi adattando il funzionamento e la configurazione del sistema alle proprie specifiche esigenze.[2]

Un sistema aperto

Entrando solo per un attimo in dettagli più tecnici, possiamo accennare al fatto che Unix è basato su un nucleo (il *kernel*) che gestisce la comunicazione *di basso livello* con la macchina, l'esecuzione dei programmi e l'uso e la condivisione o la protezione delle risorse del sistema. Questa è la parte del sistema operativo più strettamente legata all'hardware del computer. Il resto del sistema è costituito da una serie di moduli aggiuntivi finalizzati alla gestione dei vari sottosistemi che completano l'ambiente operativo (il riconoscimento degli utenti, la visualizzazione a video, l'input da tastiera, l'esecuzione dei comandi impostati dall'utente, la posta elettronica, la stampa, ecc.); questi moduli sono spesso "portabili", cioè è possibile adattarli quasi senza nessuna modifica sostanziale per poterli utilizzare su macchine completamente diverse, corredate del loro specifico *kernel*. Questo fatto contribuisce a rendere Unix un sistema operativo aperto e facilmente trasportabile su hardware diversi. È proprio per questa grande lungimiranza dei suoi progettisti iniziali che oggi possiamo disporre di una grande varietà di Unix differenti, utilizzabili su piattaforme diverse. D'altra parte la possibilità da parte di chiunque di sviluppare moduli aggiuntivi per il sistema operativo ha fatto sì che oggi non esista *uno* Unix, ma ne esistano diverse versioni che hanno finito per essere anche parzialmente incompatibili tra di loro.

Il kernel

Un sistema portabile

Le nozioni che esporremo in questa breve guida sono talmente generali che non andranno a scontrarsi con queste sottili differenze. Basti sapere che oggi esistono due standard di riferimento per gli sviluppatori di sistemi Unix: BSD, *Berkeley System Distribution*, e UNIX System V; quest'ultimo (noto anche come SVR4) è probabilmente più diffuso del BSD. Per ognuna di queste due versioni fondamentali del sistema esistono numerose implementazioni, alcune molto note e diffuse, altre conosciute ed utilizzate solo da una ristretta cerchia di utenti. A titolo di esempio possiamo dire che i sistemi Unix di tipo BSD più noti sono i sistemi operativi *open source* FreeBSD (`http://www.freebsd.org`), NetBSD (`http://www.netbsd.org`), Open BSD (`http://www.openbsd.org`), Darwin (la base Unix su cui si fonda il sistema operativo della Apple, Mac OS X;

Unix System V e BSD

[2]Per maggiori dettagli sulla storia della nascita e dell'evoluzione del sistema operativo Unix si vedano, ad esempio, le pagine del sito dell'Open Group, disponibili all'indirizzo `http://www.unix.org/what_is_unix/history_timeline.html`, oppure quelle disponibili sul sito dei Bell Labs, all'indirizzo `http://www.bell-labs.com/history/unix/`, o infine quelle, in italiano, presenti sul sito del Pluto, un gruppo di sostenitori italiani dei progetti *open source*, all'indirizzo `http://www.pluto.it/journal/pj9812/storia-unix.html`.

http://www.opendarwin.org, http://www.gnu-darwin.org, http://developer.apple.com/darwin/); sulla versione UNIX System V sono basati invece il sistema operativo Linux (http://www.linux.org/), IBM AIX (http://www.ibm.com/aix), Sun Solaris (http://www.sun.com/solaris), HP UX di Hewlett Packard (http://www.hp.com/products1/unix/operating/) e molti altri ancora.

Ci accorgeremo presto che Unix è in un certo senso un sistema operativo assai sobrio ed elegante: tutto ciò che fa di Windows un sistema coloratissimo e di grande effetto visivo, in ambiente Unix è guardato con sospetto. Qui regna la sintesi, tutto ciò che è inutile o ridondante è bandito. I messaggi del sistema vengono visualizzati solo quando è strettamente necessario: spesso un programma viene eseguito senza che l'utente legga sullo schermo nessuna comunicazione da parte della macchina; se l'esecuzione termina senza che sia stato visualizzato alcun messaggio vorrà dire che l'esecuzione del programma è andata a buon fine, mentre nel caso contrario il sistema visualizzerà un breve messaggio di errore. I comandi della *shell* sono compatti, spesso costituiti di soli due caratteri, ma hanno un'infinità di opzioni perché l'utente si deve sentire libero di modificare a proprio piacimento la modalità operativa del software che utilizza. Questo fa di Unix un sistema operativo estremamente duttile e versatile.

Come vedremo meglio nella prossima sezione, una delle caratteristiche fondamentali di Unix è quella di essere un sistema operativo *multiutente*: più persone possono usare il sistema contemporaneamente o in tempi diversi. Questo fa sì che sia necessario imparare anche un certo "galateo" che deve contraddistinguere gli utenti di un sistema multiutente, in cui le risorse sono condivise tra più persone. Il fatto stesso che lo scambio di messaggi dalla macchina all'utente sia ridotto al minimo indispensabile è un segno dell'idea che ogni operazione che possa in qualche modo appesantire il sistema, rallentando il lavoro di un altro utente, è evitata accuratamente, in modo da lasciare libere quante più risorse è possibile all'elaborazione vera e propria dei programmi.

1.2 Multiutenza e multitasking

Multitasking

Un sistema operativo *multitasking* permette di eseguire più programmi (*task*) contemporaneamente: se ad esempio viene chiesto al sistema di eseguire contemporaneamente due *processi*, A e B, la CPU eseguirà per qualche istante il processo A, poi per qualche istante il processo B, poi tornerà ad eseguire il processo A e così via. È il sistema operativo a controllare che la CPU ripartisca equamente le sue prestazioni tra tutti i processi attivi; è sempre il sistema operativo a far sì che quando un processo va in *crash*, quando si blocca in seguito al verificarsi di un errore, i restanti processi e l'intero sistema non subiscano alcun danneggiamento e possano proseguire senza conseguenze i compiti loro assegnati.

Multiutenza, account utente

Un sistema *multiutente* può essere utilizzato contemporaneamente da utenti diversi. Sotto Unix ad ogni utente del sistema viene assegnato uno *username* che lo identifica univocamente: quando si inizia una sessione di lavoro si deve "entrare" nel sistema tramite una procedura di *login* durante la quale dovremo farci riconoscere dal sistema mediante l'introduzione del nostro *username* pubblico e della nostra *password* segreta.

Dopo essere entrati nel sistema potremo lanciare i nostri processi (le applicazioni, i programmi) che verranno eseguiti in *multitasking* insieme ai processi lanciati dagli altri utenti collegati in quel momento sulla nostra stessa macchina o che hanno lasciato che il sistema proseguisse autonomamente l'esecuzione di determinati *task* anche dopo il termine della loro connessione da un terminale.

1.3 Console, terminali e terminali grafici

Può essere utile imparare a distinguere il sistema hardware mediante cui si accede alla macchina Unix su cui operiamo. Infatti, mentre quando lavoriamo con il nostro personal in ambiente Windows, siamo gli unici utenti di quel sistema, ed il computer che stiamo usando è quello che abbiamo fisicamente davanti a noi,[3] la situazione può essere molto diversa nel caso in cui si stia utilizzando un sistema Unix.

Console

La *console* è la coppia tastiera/video collegata direttamente alla macchina. In ambiente Windows la tastiera ed il monitor del nostro PC sono in un certo senso la *console* del PC stesso. Visto che però ad una stessa macchina Unix possono accedere contemporaneamente più utenti, deve essere possibile il collegamento di più tastiere e video allo stesso computer. Ed infatti di solito ad una macchina Unix sono collegati numerosi *terminali*.

Terminali alfanumerici

Un terminale è costituito da una tastiera, un video, ed una piccolissima unità di elaborazione locale, che si occupa esclusivamente di gestire la comunicazione tra il terminale stesso e l'elaboratore a cui è collegato. In sostanza il terminale si limita a visualizzare sullo schermo i messaggi del sistema e ad inviare i comandi digitati dall'utente sulla tastiera. Generalmente i terminali alfanumerici (ossia quelli che non possono visualizzare schermate grafiche, ma soltanto testuali) sono collegati al computer attraverso linee seriali, ma spesso come terminali vengono usati dei normali personal computer, dotati di un opportuno software di *emulazione di terminale*. In questo caso è del tutto ininfluente la potenza del computer e del sistema operativo della macchina utilizzata come terminale, visto che questa verrà usata esclusivamente per introdurre l'input mediante la sua tastiera e visualizzare l'output sul suo monitor, mentre ogni elaborazione vera e propria verrà eseguita dal computer (la macchina Unix) a cui il terminale è collegato.

Terminali grafici, X-terminal

Esistono dei terminali più evoluti, i cosiddetti *terminali grafici* che permettono di utilizzare un'*interfaccia grafica* (GUI) per eseguire le operazioni di input/output e quindi consentono anche di visualizzare un output di tipo grafico (immagini, disegni, grafici). I terminali grafici possono essere costituiti da apparati hardware appositamente progettati (gli *X Terminal*), dotati di un'unità di elaborazione grafica, anche molto potente, un monitor ad alta risoluzione, una tastiera ed un mouse; gli X Terminal sono generalmente connessi in rete TCP/IP mediante una interfaccia Ethernet con il sistema Unix; in questo modo possono essere utilizzati per accedere in modalità grafica a tutti i server che in rete accettano una connessione da tale terminale grafico. Una alternativa agli X Terminal veri e propri è costituita dai software denominati X Window Server che possono essere eseguiti su un personal computer (anche in un ambiente non Unix, come Windows), che sfruttando le potenzialità del personal computer, consentono di accedere in modalità grafica ad alta risoluzione alle applicazioni grafiche presenti sul sistema Unix. Infine è possibile utilizzare come terminale grafico la *console* grafica del sistema stesso (se presente) o quella di altre *workstation* Unix connesse in rete. Approfonodiremo meglio questi aspetti nel Capitolo 4.

1.4 Diritti ed attributi

Gli utenti di un sistema Unix non sono tutti uguali e soprattutto non hanno tutti gli stessi diritti. Questa affermazione un po' perentoria potrebbe suonare male per qualche sincero democratico e far pensare che Unix sia un sistema operativo illiberale. Naturalmente nulla potrebbe essere più falso: Unix garantisce la libertà di azione di tutti gli utenti facendo in modo che nessun altro utente non autorizzato

[3] Trascuriamo per semplicità la possibilità che più utenti contemporanei possano accedere tramite i *terminal services* ad uno stesso sistema in ambiente Microsoft Windows Server.

possa in qualche modo violare la nostra privacy o distruggere o manomettere le nostre informazioni. In questo modo viene anche garantita una certa sicurezza dell'intero sistema: nessun utente "normale" potrà manometterlo compromettendone il corretto funzionamento; ma non solo: nessun utente "normale" potrà commettere errori talmente gravi nell'uso del sistema tanto da danneggiare altri utenti o il sistema stesso.

Cosa vuol dire più esattamente tutto questo? Abbiamo visto che ogni utente è identificato univocamente all'interno del sistema mediante uno *username*. Gli utenti del sistema sono distribuiti in più *gruppi*; ogni utente fa parte almeno di un gruppo. Ad esempio nel sistema Unix del Dipartimento di Matematica dell'Università "La Sapienza" di Roma, gli utenti sono stati divisi in diversi gruppi, denominati ad esempio "teograf", che raccoglie coloro che si occupano di Teoria dei Grafi, "algegeo", che raccoglie coloro che si occupano di Algebra e Geometria, e così via. Gli utenti possono essere raggruppati in modo tale da attribuire solo ad alcuni di loro (quelli facenti parte di un gruppo specifico) la capacità di effettuare determinate operazioni sul sistema stesso (ad esempio controllare la coda dei messaggi di posta elettronica in uscita dal sistema o compiere altre attività di gestione del sistema stesso).

Gruppi di utenti

Esiste poi un utente privilegiato, il cui *username* è *root*, che viene assegnato all'amministratore del sistema, il cosiddetto *system manager*. Questi è una figura assai importante nella gestione di un sistema Unix. È colui che può modificare la configurazione dell'intero sistema e che ha la possibilità di verificare ogni cosa all'interno del sistema stesso. È il caso di dire che è anche l'unico utente che possa combinare dei guai seri su una macchina Unix![4]

Root, l'amministratore del sistema

Come si traduce tutto questo in pratica? Innanzi tutto ad ogni utente viene assegnata una propria *home directory* nel *filesystem* della macchina. Ad utenti differenti corrispondono *home directory* diverse; tali directory sono di proprietà degli utenti a cui sono assegnate e di solito hanno lo stesso nome dell'utente. Anche ogni altro file o directory nel *filesystem* ha un proprietario che, mediante un apposito comando, ne stabilisce le modalità d'uso (i diritti di accesso) per se stesso e per gli altri utenti del sistema. È possibile, ad esempio, fare in modo che un certo file possa essere modificato solo dal proprietario, che possano leggerlo tutti gli utenti del gruppo del proprietario e che gli altri utenti non possano né leggerlo, né modificarlo o cancellarlo. Lo stesso è possibile fare con le directory, di cui si possono stabilire i diritti di lettura, scrittura ed accesso, e per i *file binari* contenenti le applicazioni (i programmi), di cui è possibile stabilire i diritti di lettura, scrittura ed esecuzione.

Home directory personali

Permessi di accesso a file e directory

L'utente *root* non ha questi vincoli, ma può accedere in qualsiasi modo a qualunque file o directory presente nel *filesystem*, a prescindere dagli attributi di protezione attivati dagli utenti.

È bene tenere presente che, come vedremo meglio in seguito, ogni operazione eseguita su una macchina Unix viene effettuata a nome e per conto di un determinato utente. Non esistono *task* o programmi che girano in modalità anonima: ogni programma viene eseguito per conto di un determinato utente e pertanto ne acquisisce tutti i permessi ed i vincoli.

Ownership dei processi

1.5 Uno sguardo al filesystem

Abbiamo usato più volte il termine *filesystem*, ma cosa significa esattamente? Con questa parola si indica l'insieme dei supporti di memorizzazione, fisici o virtuali,

[4] Forse è bene precisare che sebbene i file di sistema e le directory degli altri utenti siano generalmente ben protette, è difficile mettere al riparo l'utente dalla possibilità di compiere danni sui propri file. Lavorando con la *shell* Unix non esistono possibilità di annullare l'effetto dei comandi impartiti al sistema (*undo*) o di recuperare file temporaneamente spostati nel "cestino"...
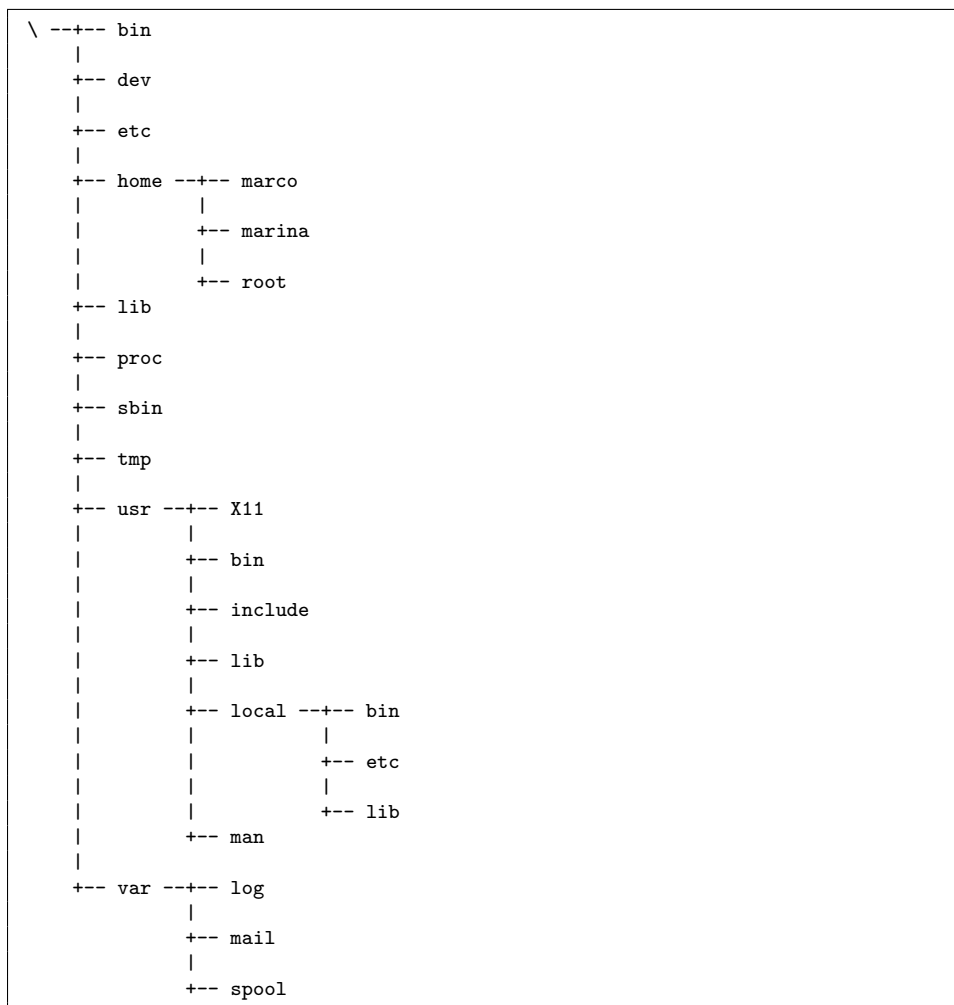
Figura 1.1: Un esempio di struttura del *filesystem* di un sistema Unix

collegati al sistema (in gergo: *montati* sul sistema). Chi ha avuto modo di usare un PC in ambiente Windows sa bene che ogni unità a disco è identificata da una lettera dell'alfabeto: A è il dischetto magnetico, C è il primo disco rigido, D è il secondo disco rigido e così via. In ambiente Unix la situazione cambia radicalmente. Esiste una unità (un disco) principale (*root*, radice, da non confondersi però con lo *username* del *system manager*) a cui vengono "agganciate" come sottodirectory tutte le altre unità, siano esse diversi tipi di *hard disk* o di unità di memorizzazione di massa presenti sul computer stesso, o "volumi" messi in condivisione via rete da altri computer attraverso appositi protocolli. L'insieme di tutte le unità di memorizzazione (chiamate "volumi"), accorpate in un'unica grande struttura ad albero, costituiscono il *filesystem*.

Struttura ad albero del filesystem

Generalmente, su quasi ogni sistema Unix, sono presenti alcune directory che rivestono una certa importanza all'interno del sistema e che hanno quasi sempre lo stesso nome. A titolo di esempio consideriamo la struttura ad albero riportata in Figura 1.1 che rappresenta parte di un ipotetico *filesystem* (assai ridotto, per la verità).

Diamo una rapidissima scorsa al contenuto delle directory elencate in figura:

`/bin` Contiene molti dei file binari (eseguibili) presenti sul sistema.

`/dev` È una directory molto importante che contiene i puntatori ai *device driver* delle unità hardware installate sul sistema. Sono alcune di quelle estensioni del *kernel* di cui parlavamo nelle pagine precedenti, che permettono al sistema di gestire le unità ad esso collegate. Ad esempio il file `/dev/ttyS0` gestisce l'input/output attraverso il primo terminale collegato al sistema, mentre `/dev/console` gestisce la *console* del sistema. `/dev/null` è l'unità "nulla", che risulta assai utile in alcune situazioni, come vedremo in seguito.

`/etc` Contiene una serie di file che non trovano collocazione migliore in altre directory; sono per lo più file di configurazione generale del sistema.

`/home` Contiene le *home directory* degli utenti.

`/lib` Contiene le "librerie" di sistema, dei file che contengono parte di codice eseguibile e che vengono utilizzati per la compilazione di applicazioni. Questo consente di ridurre la dimensione dei programmi, inserendo nelle librerie parti di codice comuni a più applicazioni.

`/proc` È una directory piuttosto particolare: i file che contiene non sono memorizzati su disco, ma direttamente nella memoria dell'elaboratore; contengono i riferimenti ai vari processi attivi nel sistema e le informazioni utili per potervi accedere.

`/sbin` Contiene i file eseguibili ("binari") riservati all'amministratore del sistema.

`/tmp` È la directory temporanea di default. Spesso le applicazioni devono scrivere dei dati su un file temporaneo, che al termine dell'esecuzione verrà cancellato; in questi casi spesso usano la directory `/tmp`, che è sempre presente sui sistemi Unix.

`/usr` Contiene numerose sottodirectory molto importanti per il sistema: nulla di ciò che è contenuto sotto la directory `/usr` è di vitale importanza per il funzionamento della macchina, ma spesso è proprio sotto `/usr` che vengono collocate tutte quelle cose che rendono utile il sistema.

`/usr/X11` Contiene ciò che riguarda l'interfaccia grafica X Window.

`/usr/bin` Altri eseguibili (file binari).

`/usr/include` Contiene i file *include* per i programmi in linguaggio C relativi alle librerie installate sul sistema.

`/usr/lib` Altre librerie di sistema.

`/usr/local` Contiene file tipici della nostra macchina; tipicamente applicazioni installate successivamente al sistema operativo.

`/usr/man` Contiene le *pagine di manuale* (l'*help on-line* del sistema operativo Unix).

`/var` Contiene diversi tipi di file il cui contenuto varia con una frequenza elevata; i file presenti in queste directory vengono gestiti da altri programmi attivi sul sistema.

`/var/log` Contiene il "registro storico" (i *file di log*) degli eventi accaduti sul sistema e tracciati da alcuni dei programmi attivi sul sistema stesso.

`/var/mail` Contiene i file con le *mailbox* di posta elettronica degli utenti del sistema, con i messaggi di posta elettronica giacenti e non ancora letti dai rispettivi destinatari.

`/var/spool` Contiene i file in attesa di essere elaborati da altri programmi, come ad esempio la coda di stampa e la coda dei messaggi di posta elettronica in uscita.

Figura 1.2: Silicon Graphics Indy

Capitolo 2

Comandi fondamentali

In questo capitolo diamo un rapido sguardo ai comandi principali della *shell* del sistema operativo Unix. Questi comandi ci consentono di eseguire programmi, di spostarci nelle directory del *filesystem* e di gestire facilmente i file e le funzioni di base del sistema.

Al di là delle funzionalità semplificate offerte da alcuni *desktop manager* in ambiente grafico, il modo migliore per interagire con un sistema Unix è attraverso una *shell*. È un programma che interpreta ed esegue i comandi immessi dall'utente sulla base di un linguaggio che descriveremo per grandi linee nelle pagine seguenti. La *shell* in genere viene eseguita in modalità interattiva, ossia in una modalità secondo cui la *shell* attende che l'utente immetta un comando, poi lo interpreta e lo esegue e ritorna nello stato di attesa per il comando successivo; tuttavia la *shell* può essere utilizzata anche come interprete per l'esecuzione di un intero programma scritto nel linguaggio della stessa *shell*: un programma per la *shell* sarà quindi una sequenza di comandi e di istruzioni in grado di controllare e di modificare l'ordine con cui tali comandi devono essere eseguiti. Un programma per la *shell* viene chiamato in gergo uno *script* (o *shell script*).

La shell dei comandi

La *shell* è il programma che di solito viene eseguito automaticamente dal sistema quando un utente effettua il *login*; questa specifica istanza della *shell* viene quindi chiamata *shell di login*.

Volendo fare un parallelo con il sistema operativo Windows, possiamo dire che la *shell* svolge più o meno la stessa funzione del programma `command.com` o di `cmd.exe`, ma è molto più potente e versatile. Gli *shell script* sono quindi l'analogo dei file *batch* (i file "`*.bat`") in ambiente Windows.

Principali shell di comandi Unix

Esistono diverse *shell* sotto Unix: tra queste citiamo la Bourne Shell (scritta da Stephen Bourne), che può essere richiamata con il comando `sh`, la `bash`, o Bourne Again Shell, un'evoluzione di `sh`, la Korn Shell `ksh`, la C Shell `csh`, molto amata dai programmatori in quanto ha una sintassi simile a quella del linguaggio C, la `tcsh`, evoluzione della C Shell, ed altre ancora. In effetti le differenze tra queste *shell* sono numerose, ma risultano evidenti soprattutto scrivendo degli *script*, perché per il resto sono abbastanza simili fra loro.

È importante osservare che in ambiente Unix le lettere maiuscole e minuscole sono completamente diverse: sotto Windows scrivendo `DIR` e `dir` ci si riferisce allo stesso comando, come pure è possibile riferirsi ad un certo file chiamandolo `pippo.txt` o `PIPPO.TXT` indifferentemente. Sotto Unix non vale la stessa cosa: i nomi dei comandi sono *case sensitive* e devono essere digitati rispettando le lettere maiuscole e minuscole e lo stesso vale per i nomi dei file e delle directory. Inoltre non esistono limitazioni sui nomi dei file: questi possono essere anche molto lunghi e non è necessaria l'"estensione"; di fatto il simbolo "`.`" è un carattere come ogni altro nel nome del file (ma ha una funzione particolare se utilizzato come primo carattere

del nome di un file, come vedremo tra breve) ed anche il carattere di spaziatura può far parte del nome del file, anche se personalmente sconsiglio di utilizzarlo perché ci obbliga ad usare delle accortezze ulteriori per riferirci successivamente a quel file. Ad esempio potremmo chiamare il file che contiene il testo di questo capitolo "`guida unix.capitolo-2`".

2.1 Accesso al sistema e chiusura della sessione

Abbiamo già detto che per accedere alla macchina si deve effettuare una procedura di "riconoscimento" detta *login*. Il sistema ci chiede innanzi tutto di inserire il nostro *username* e poi la *password* segreta. Per capire meglio vediamo un esempio:

```
woodstock login: marco
Password:
Last login: Fri Apr 21 10:27:08 on ttyS2
$ _
```

La *password* non viene visualizzata a video quando la inseriamo, per impedire a chi ci sta guardando di scoprirla e poter quindi accedere al sistema a nome nostro; al contrario lo *username* è la componente pubblica del nostro *account* di utenti del sistema e serve agli altri per poter comunicare con noi, come vedremo nelle pagine seguenti. Il messaggio "Last login..." ci comunica la data e l'ora dell'ultima volta che siamo entrati nel sistema ed il nome del terminale da cui ci siamo collegati, in questo caso il terminale `ttyS2`. Questo è un messaggio utile, perché ci permette di controllare se per caso ci sono state intrusioni fraudolente a nostro nome sul sistema, da parte di qualcuno che è riuscito a scoprire la nostra *password*.

Prompt della linea di comando

Il simbolo "`$`" che viene visualizzato davanti al cursore è il *prompt*, ossia l'indicazione che la *shell* è pronta ad accettare un nostro comando immesso da tastiera; quello riportato in queste pagine è solo un esempio: il *prompt* potrebbe variare da sistema a sistema o essere configurato diversamente dall'utente (vedremo più avanti come è possibile modificare il *prompt*). Tipicamente l'utente *root* ha un *prompt* diverso da quello degli altri utenti, proprio per evidenziare all'operatore che sta utilizzando comandi che saranno eseguiti con i privilegi di *root* e non con quelli del suo account personale: commettere un errore, anche solo per una semplice distrazione, potrebbe avere effetti disastrosi sull'integrità del sistema. Negli esempi di questa guida indicheremo con "`#`" il *prompt* di *root*.

Modifica della password

Per cambiare la *password* si deve digitare il comando `passwd`: il sistema chiederà di inserire prima la vecchia *password* e poi, se questa è corretta, chiederà per due volte (per sicurezza) di inserire la nuova *password* che l'utente intende impostare per proteggere il proprio *account*. Gli *username* e le *password* cifrate, insieme ad altre informazioni di base sugli utenti accreditati sul sistema, sono memorizzate nel file `/etc/passwd`, che tutti gli utenti possono visualizzare. Su alcuni sistemi, per maggiore sicurezza, le *password* sono memorizzate invece nel file `/etc/shadow` (sempre in forma cifrata) e tale file è leggibile solo dall'utente *root*.

Chiusura della sessione di lavoro, logout, exit

Quando abbiamo finito di lavorare, prima di andarcene, dobbiamo scollegarci, effettuare cioè la procedura di *logout* che serve proprio per comunicare al sistema che abbiamo terminato di lavorare e che quindi può chiudere la sessione rimettendosi in attesa del prossimo utente. Per scollegarci possiamo usare il comando `exit`, o il comando `logout` se stiamo lasciando la *shell* di login; in alcune *shell* anche la sequenza di tasti Ctrl-d (che corrisponde al carattere di "fine-file") produce lo stesso effetto. Effettuato il *logout* il sistema presenterà nuovamente il messaggio di login in attesa che qualcun'altro si colleghi dallo stesso terminale.

Di norma nei CED e nei centri di calcolo i terminali vengono lasciati sempre accesi, ma, salvo esplicita controindicazione, è possibile anche spegnerli dopo che si è effettuato il *logout*. È fondamentale invece non spegnere mai la macchina che ospita il sistema: solo *root* può effettuare la procedura di *shutdown* al termine della quale sarà possibile spegnere il computer. Non si deve mai spegnere un sistema Unix senza prima aver completato la procedura di *shutdown*, pena la perdita irreparabile dei dati del *filesystem*. Di seguito riportiamo un frammento della sequenza di *shutdown* di una macchina in ambiente Linux: durante questa fase il sistema operativo invia il segnale di terminazione a tutti i processi attivi, in modo che questi possano concludere l'esecuzione chiudendo correttamente eventuali file aperti.

Shutdown

```
# shutdown -h now
Broadcast message from root (pts/1) Tue Aug 30 17:03:16 2005...
The system is going down for system halt NOW !!
...
System halted.
```

2.2 Muoversi nel filesystem

Appena entrati nel sistema ci troviamo all'interno della nostra *home directory*. Per verificare il nome della directory possiamo usare il comando `pwd` (*print working directory*).

Directory corrente, pwd

```
$ pwd
/home/marco
$ _
```

Il simbolo "/" (*slash*), oltre ad indicare la *root directory* del *filesystem*, è utilizzato come separatore tra le directory che fanno parte di uno stesso *path*; sui sistemi Windows la stessa funzione è svolta dal carattere "\" (*backslash*).

Di sicuro la prima cosa che ci viene in mente è quella di visualizzare il contenuto della nostra directory. Per far questo si deve usare il comando `ls` (*list*), equivalente al comando `dir` del *prompt* dei comandi di Windows. Ad esempio potremmo ottenere il seguente output:

Lista dei file, ls

```
$ ls
lettera     mail      pippo.c      progetto    tesi
libro       pippo     pippo.zip    src
$ _
```

Vengono visualizzati nove nomi, ma non è chiaro se siano nomi di file di dati, di sottodirectory o di programmi eseguibili. Abbiamo accennato precedentemente alle numerose opzioni che generalmente consentono di modificare opportunamente l'esecuzione di un certo comando o programa; in generale una riga di comando è costituita dal comando stesso seguito da una sequenza di opzioni indicate mediante caratteri (o sequenze di caratteri) preceduti dal simbolo "-", e da parametri che vengono passati al comando. Le opzioni consentono di modificare in parte il modo con cui il comando viene eseguito. Ad esempio potremmo provare la seguente variante del comando `ls`:

Opzioni e parametri

```
$ ls -F /home/marco
lettera     mail/     pippo.c      progetto@   tesi/
libro/      pippo*    pippo.zip    src/
$ _
```

Il comando è "ls", la sequenza "-F" rappresenta un'opzione del comando, mentre "/home/marco" è il parametro passato al comando stesso. L'opzione "-F" del comando ls fa sì che accanto ad alcuni nomi di file venga visualizzato un simbolo che non fa parte del nome, ma che serve ad indicare di che tipo di file si tratta: lo *slash* "/" indica che è una directory, l'asterisco "*" indica che si tratta di un file eseguibile, mentre la chiocciola "@" sta ad indicare che quel file (o directory) non è fisicamente presente nella nostra directory, ma è un *link*, un rimando, un collegamento, ad un file (o ad una directory) che si trova da un'altra parte nel *filesystem*.

Lista di file in formato "lungo"

Si ottiene un'altra versione del comando ls aggiungendo l'opzione "-l" (*long*); visto che è possibile specificare più opzioni su uno stesso comando, vediamo quale potrebbe essere l'output del comando "ls -lF" (che equivale anche ad "ls -l -F", visto che in molti casi le opzioni dei comandi possono essere "raggruppate"):

```
$ ls -lF
-rw-r--r--  1 marco users    937 Apr 23 12:43 lettera
drwxr-xr-x  2 marco users   1024 Apr 10 16:04 libro/
drwx------  2 marco users   1024 Feb 01 09:32 mail/
-rwxr-x---  1 marco users  37513 Mar 10 11:55 pippo*
-rw-r--r--  1 marco users  18722 Mar 10 11:30 pippo.c
-rw-r--r--  1 marco users  23946 Mar 10 12:03 pippo.zip
lrwxrwxr--  1 marco users      8 Apr 04 18:16 progetto -> /usr/proj
drwxrwx---  2 marco users   1024 Mar 10 08:47 src/
drwxr--r--  2 marco users   1024 Feb 12  1995 tesi/
$ _
```

Illustriamo brevemente il significato delle numerose informazioni presentate dal comando "ls -l". Il primo carattere di ogni riga può essere "d" per indicare che si tratta di una directory, "l" per indicare un link o "-" per indicare che si tratta di un normale file. I successivi nove caratteri rappresentano in forma sintetica i permessi di accesso assegnati ai file; devono essere letti raggruppandoli a tre a tre. I primi tre caratteri indicano i diritti del proprietario di tale file sul file stesso; i successivi tre caratteri indicano i diritti degli altri utenti facenti parte del gruppo proprietario del file, mentre gli ultimi tre caratteri rappresentano i diritti di tutti gli altri utenti del sistema. Una "x" sta ad indicare il diritto di esecuzione di tale file (mentre è chiaro cosa significa eseguire un programma, è opportuno chiarire che "eseguire" una directory significa poterci entrare dentro). Il carattere "r" sta ad indicare il diritto di leggere tale file (*read*), mentre "w" indica il diritto di poterci scrivere sopra (*write*), modificandolo o cancellandolo.

Permessi di accesso ai file

Di seguito viene riportato lo *username* del proprietario del file (es.: marco) ed il nome del gruppo di appartenenza (es.: users). Viene poi visualizzata la dimensione del file, la data e l'ora in cui è stato creato (o modificato per l'ultima volta) ed infine il nome.

Nell'esempio il file pippo.zip può essere letto e modificato dal proprietario (marco), mentre può essere soltanto letto dagli altri utenti del sistema; è stato creato il 10 Marzo dell'anno in corso alle 12:03. Il file progetto è un link alla directory /usr/proj e può essere utilizzato in lettura, scrittura ed esecuzione solo dal proprietario e dagli utenti del gruppo users, ma non dagli altri utenti del sistema che possono solo accedervi in lettura.

Delle numerosissime opzioni del comando ls ci limitiamo ad illustrarne solo un'altra, che ci permette di introdurre anche qualche interessante novità a proposito dei nomi dei file; cominciamo con il solito esempio, osservando l'output prodotto dal comando "ls -a":

```
$ ls -aF
./           .bashrc    lettera     pippo*      progetto@
../          .exrc      libro/      pippo.c     src/
.Xdefaults   .newsrc    mail/       pippo.zip   tesi/
$ _
```

Da dove spuntano questi strani file preceduti da un punto e perché fino ad ora non li avevamo visti? Per convenzione sotto Unix (ma anche in ambiente Windows) il nome "`.`" (punto) sta ad indicare la directory corrente, mentre con "`..`" si indica la directory "padre" nell'albero del *filesystem*[1]. Gli altri file preceduti dal punto sono dei normali file (o directory, ma non nel nostro caso) che però non vengono visualizzati dal comando `ls` proprio perché hanno un nome che comincia con il punto. In questo modo possiamo evitare di visualizzare alcuni file che devono essere presenti nella nostra directory (ad esempio i file di configurazione), ma che non utilizzeremo direttamente quasi mai e dei quali dimenticheremo presto la presenza. Visualizzarli ogni volta renderebbe l'output del comando `ls` eccessivamente ridondante e quindi tali file vengono, in un certo senso, "nascosti".

Nomi di file che iniziano con il punto

Per spostarsi attraverso le directory del *filesystem* si usa il comando `cd` (*change directory*). Ad esempio per "scendere" nella directory `src` si dovrà digitare il comando "`cd src`", mentre per risalire nella directory padre (la nostra *home directory*) dovremo digitare il comando "`cd ..`". A proposito della *home directory* è utile osservare che ci si può riferire ad essa mediante l'uso del simbolo "`~`" (tilde), mentre per riferirsi alla *home directory* dell'utente "`marina`" si potrà usare la stringa "`~marina`" (ad esempio si potrà digitare "`cd ~marina`" per spostarci nella sua directory: questo modo di impostare il comando è sicuramente più sintetico del canonico "`cd /home/marina`"). Visto che si presenta frequentemente la necessità di rientrare nella propria *home directory*, il comando `cd` dato senza nessun parametro assolve efficacemente tale scopo.

Cambiare directory corrente, cd

2.3 Gestione di file e directory

Abbiamo visto nella sezione precedente che ad ogni file sono associati degli attributi per regolare l'accesso da parte degli utenti del sistema. Per modificare gli *attributi* di un file si deve utilizzare il comando `chmod`, che ha la seguente sintassi: "`chmod` *attributi file*".

Modifica dei permessi dei file, chmod

Il parametro *attributi* può avere varie forme, ma forse la più semplice di tutte è quella numerica, secondo la seguente regoletta. Si associa valore 100 al diritto di eseguire il file per il proprietario, 10 per gli utenti del gruppo e 1 per tutti gli altri; si associa 200 al diritto di accesso in scrittura al file per il proprietario, 20 per gli utenti del gruppo e 2 per tutti gli altri; infine si associa 400 al diritto di accesso in lettura al file per il proprietario, 40 per gli utenti del gruppo e 4 per gli altri. Sommando questi numeri si ottiene l'attributo da assegnare a quel file. Così, ad esempio, il parametro `700` equivale a stabilire che l'unico a poter leggere, scrivere ed eseguire il file è il proprietario ($100 + 200 + 400 = 700$), mentre con `644` si indica il fatto che il file può essere letto e modificato dal proprietario, ma può essere soltanto letto dagli altri utenti; in quest'ultimo caso, volendo associare tale attributo al file `pippo.c` daremo il comando

```
$ chmod 644 pippo.c
$ _
```

[1] Riferendoci all'esempio riportato a pagina 6, possiamo dire che `/usr` è la directory padre di `/usr/bin` e che quest'ultima è la directory "figlio" di `/usr`.

Creazione e cancellazione di directory, mkdir, rmdir

Per creare una directory si usa il comando `mkdir` (*make directory*); ad esempio con il comando

```
$ mkdir esempio
$ _
```

possiamo creare una nuova directory nella directory corrente.

Per rimuovere una directory vuota (che non contiene alcun file) si deve usare invece il comando `rmdir` (*remove directory*); ad esempio:

```
$ rmdir esempio
$ _
```

Il comando viene prontamente eseguito dal sistema senza chiederci ulteriore conferma. La directory non viene cancellata se contiene al suo interno dei file o delle sottodirectory (è necessario eliminare prima tali file e directory per poter poi cancellare la directory che li conteneva).

Copia e spostamento di file, cp, mv

Per creare una copia di un file si deve usare il comando `cp` (*copy*); ad esempio volendo copiare il file `pippo.c` nella directory `src` dovremo dare il comando:

```
$ cp pippo.c src
$ _
```

Sintassi analoga ha anche il comando `mv` (*move*) che serve a spostare i file da una directory all'altra, ad esempio per spostare tutti i file il cui nome termina con "`.c`" dalla directory `src` alla directory `tesi` potremo dare il seguente comando:

```
$ mv src/*.c tesi
$ _
```

Possiamo utilizzare il comando `mv` anche per cambiare il nome ad un file, ad esempio se volessimo modificare il nome del file `pippo.c` e chiamarlo `pluto.c`, potremmo usare il comando:

```
$ mv pippo.c pluto.c
$ _
```

Cancellazione di file e directory, rm

Infine per cancellare un file si usa il comando `rm` (*remove*); si deve porre particolare attenzione nell'uso di questo comando, perché se non si specifica l'opzione "`-i`" (*interactive*), il sistema eseguirà la cancellazione dei file senza chiedere ulteriori conferme. Ad esempio il seguente comando cancella ogni file contenuto nella directory `tesi`:

```
$ rm tesi/*
$ _
```

Usando l'opzione "`-i`" il sistema ci chiede conferma dell'operazione; alla domanda del sistema si deve rispondere con "`y`" (*yes*) oppure con "`n`" (*no*):

```
$ rm -i pippo.c
rm: remove 'pippo.c'? y
$ _
```

Con l'opzione "-r" il comando rm può essere utilizzato anche per cancellare "ricorsivamente" una directory con tutti i file e le sottodirectory in essa contenuti; naturalmente i file e le directory su cui l'utente non ha permesso di accesso in scrittura non potranno essere rimossi:

```
$ rm -r src
$ _
```

Per sapere quanto spazio occupa una certa directory o ogni singolo file presente in una directory, si può usare il comando du (*disk usage*); ad esempio il comando "du -sk *" visualizza il numero di Kbyte occupati da ogni sigolo file presente nella directory corrente. Al contrario per sapere quanto spazio è ancora libero in ogni singola partizione del *filesystem* si può utilizzare il comando df (*disk free*); il seguente comando visualizza lo spazio libero ed occupato su ogni partizione in Kbyte:

Spazio libero e occupato sul filesystem, du, df

```
$ df -k
Filesystem           kbytes    used    avail capacity  Mounted on
/dev/dsk/c0t0d0s0    245679  111360   109752    51%    /
/dev/dsk/c0t0d0s6   3098743 1916210  1120559    64%    /usr
/proc                     0       0        0     0%    /proc
/dev/dsk/c0t0d0s5   2055463 1289267   704533    65%    /var
swap                1138744      64  1138680     1%    /tmp
/dev/dsk/c0t0d0s7  32016550 5201840 26494545    17%    /export
$ _
```

Per individuare la collocazione di un file o una directory nel *filesystem* si può utilizzare il comando find che consente di cercare uno o più file sulla base del nome o di un pattern che deve corrispondere a quello dei nomi dei file cercati. Il comando è piuttosto sofisticato e consente di effettuare numerose operazioni sui file trovati. Per visualizzare semplicemente il *path* dei file individuati si può usare la seguente sintassi: "find *path* -name *pattern* -print". I parametri *path* e *pattern* indicano rispettivamente il percorso della directory radice del sottoalbero del *filesystem* in cui si vuole eseguire la ricerca (indicheremo "/" se vogliamo compiere una ricerca su tutto il *filesystem*) e il pattern (descritto con un'espressione regolare) a cui deve corrispondere il nome dei file che stiamo cercando (ad esempio "pippo.c" se cerchiamo esattamente un file con quel nome, oppure "es*.c" se cerchiamo tutti i file il cui nome inizia con "es" e termina con ".c"). Ad esempio per cercare tutti i file il cui nome termina con ".txt" nella directory corrente (e nelle sue sottodirectory), possiamo usare il seguente comando:

Ricerca di file e directory, find

```
$ find . -name "*.txt" -print
./src/dati.txt
./relazione/tmp/lettera.txt
$_
```

2.4 Visualizzazione e stampa di file

È molto utile poter visualizzare e stampare il contenuto di un file di testo. Ancora più utile è poterne creare di nuovi e modificarne il contenuto. Anche perché per la configurazione delle caratteristiche fondamentali di un sistema Unix o anche soltanto dell'ambiente di lavoro di un singolo utente, è necessario modificare il contenuto di alcuni file di configurazione, costituiti appunto da semplici file di testo che contengono alcune istruzioni in un formato ben noto e documentato (Unix è un sistema

operativo aperto!). Dunque visualizzare, stampare e poi anche creare e modificare il contenuto di un file sono operazioni che qualunque utente di un sistema Unix deve saper compiere con una certa agilità. A questo scopo sono stati implementati diversi comandi: in questa sezione ne vedremo solo alcuni, quelli che riteniamo veramente indispensabili, rimandando al prossimo Capitolo 3 la descrizione degli editor principali.

Visualizzazione di file ASCII, cat, more, less, view

Il primo comando, ed anche il più elementare, è `cat`, che serve per concatenare due o più file, ma che può essere usato anche per visualizzare il contenuto di un file di testo. Ad esempio il seguente comando visualizza sul terminale il contenuto del file `pippo.c` (un banale programmino in linguaggio C):

```
$ cat src/ciao.c
#include <stdlib.h>
#include <stdio.h>
int main(void) {
  printf("Ciao!\n");
  return(1);
}
$ _
```

Se il file è molto lungo e non può essere visualizzato all'interno di una schermata del terminale, l'output scorrerà sullo schermo fino all'ultima riga del file, senza che sia possibile leggerne il contenuto. Per ovviare a questo problema si usa il comando `more`, che, come `cat`, visualizza il contenuto del file sullo schermo, ma alla fine di ogni schermata rimane in attesa che l'utente batta un tasto prima di visualizzare la schermata successiva. Il comando `less` è una evoluzione di `more` ed offre qualche funzionalità in più per scorrere le righe del file; su alcuni sistemi `more` è un *alias* di `less`. I comandi interattivi `more` e `less` accettano diversi tasti per scorrere il contenuto del file visualizzato:

[Spazio] l'output scorre in avanti di una schermata;

[Return] l'output scorre in avanti di una sola riga;

[b] l'output scorre indietro (*back*) di una schermata;

[q] il comando `more` viene interrotto (*quit*);

[G] va alla fine del file (solo `less`);

[g] va all'inizio del file (solo `less`);

[v] richiama l'editor di default (quello specificato nella variabile di ambiente EDITOR, tipicamente l'editor "vi") per modificare il file;

[/] effettua una ricerca sull'intero file per individuare la stringa digitata subito dopo aver battuto il tasto [/].

Il comando `more` può anche essere utilizzato per filtrare verso il terminale l'output proveniente da un altro comando, attraverso la possibilità di effettuare il *piping* dei comandi messa a disposizione dalla *shell*. Ad esempio se l'output del comando "`ls -l`" dovesse essere troppo lungo potremmo utilizzare il comando "`ls -l | more`" che prima di inviare l'output al terminale lo filtra attraverso `more`, visualizzandolo una schermata alla volta.

Per visualizzare in modo più confortevole il contenuto di un file, si può utilizzare il comando `view` (o anche "`vi -R`") che consente di scorrere il contenuto di un

file di testo utilizzando i tipici tasti di spostamento del cursore; per terminare la visualizzazione del file con `view` è necessario digitare la sequenza "`:q`".

Visualizzare sullo schermo del terminale il contenuto di un file che non contenga del semplice testo composto da caratteri ASCII standard (ad esempio un file binario o un file che contenga informazioni codificate in un altro modo) potrebbe produrre l'invio al terminale di sequenze di caratteri in grado di compromettere la sessione di lavoro sul terminale stesso. È bene quindi assicurarsi della natura del contenuto di un file prima di visualizzarlo sullo schermo con `cat` o altri comandi analoghi. Per far questo si può utilizzare il comando `file` che visualizza in output il tipo di dati contenuti nel file il cui nome gli è stato passato come argomento; ad esempio:

Riconoscere il tipo di un file, file

```
$ file src/ciao.c
src/ciao.c: ASCII C program text
$ file /bin/ls
/bin/ls: ELF 32-bit MSB executable SPARC Version 1, dynamically
linked, stripped
$ _
```

Nel primo caso (il file `ciao.c`) si tratta di un file di testo ASCII il cui contenuto può essere visualizzato correttamente con `cat`, mentre nel secondo caso (il file `/bin/ls` che contiene l'eseguibile del comando `ls`) si tratta di un file binario che non può essere visualizzato con il comando `cat` o uno degli altri comandi visti poc'anzi.

I comandi `head` e `tail` consentono rispettivamente di visualizzare la parte iniziale e la parte finale di un file di testo ASCII. Di default entrambi visualizzano le prime o le ultime 10 righe del file, ma con l'opzione "-n" si può estendere o ridurre la selezione ad n righe. Inoltre il comando `tail` ammette anche l'opzione "`-f`" che consente di tenere aperto un certo file e di visualizzare le ultime righe man mano che queste vengono aggiunte al file stesso; è un comando utilissimo per tenere sotto controllo il contenuto di un file di log o di un file di dati prodotto da un programma in esecuzione; ad esempio per visualizzare in tempo reale il file di log principale del sistema si può usare il comando "`tail -f /var/log/syslog`" (per terminare il comando si deve battere Ctrl-c).

Visualizzare l'inizio e la fine di un file, head, tail

Il comando `wc` (*word count*) consente di contare i caratteri, le parole o le righe contenute in un certo file; ad esempio per contare il numero di account utente presenti sul sistema si può usare il seguente comando:

Conteggio di caratteri, parole e righe, wc

```
$ wc -l /etc/passwd
    398 /etc/passwd
$ _
```

L'uso della stampante è una tipica operazione "*site dependent*", che varia a seconda del tipo di stampante e dei cosiddetti *filtri di stampa* installati sul sistema.

La differenza sostanziale è tra file di testo (scritti con un normale editor, senza fare uso di particolari caratteri di controllo o di formattazione del testo) che contengono solo caratteri ASCII, e file PostScript (che di solito sono riconoscibili perché il nome termina con "`.ps`") che contengono una rappresentazione di livello "tipografico" di un documento o di una immagine grafica. Se la stampante è una stampante generica sarà più semplice stampare i primi, se è una stampante PostScript sarà invece assai semplice stampare il secondo tipo di file. In ogni caso, come per quasi ogni altra operazione sotto Unix, nulla è impossibile e quindi saremo anche in grado di stampare normali file di testo su stampanti Postscript e file Postscript su stampanti generiche.

Il comando fondamentale in questo caso è `lpr`. Come `more` anche questo può essere usato sia come comando che come filtro per l'output di un altro programma.

Stampare un file, lpr, ghostview

Così ad esempio i due comandi "`lpr pippo.c`" e "`cat pippo.c | lpr`" sono assolutamente equivalenti e producono entrambi la stampa del file `pippo.c`. In questo caso abbiamo inviato alla *coda di stampa* un file di testo che potrà essere stampato correttamente se il sistema dispone di una stampante generica o se sono stati predisposti degli opportuni filtri automatici per la stampante PostScript; in modo del tutto analogo possiamo stampare il file PostScript "`tesi/cap1.ps`", ad esempio con il comando "`lpr tesi/cap1.ps`", se la nostra stampante è una unità PostScript o se il sistema dispone di un opportuno filtro di conversione per il formato PostScript (ad esempio `gs`, anche noto come *GhostScript*). Se si dispone di un terminale grafico X Window un modo molto comodo per visualizzare e stampare un file PostScript è quello di utilizzare il programma `ghostview`, che fornisce una semplice interfaccia per GhostScript, guidata mediante l'uso dei menù.

Gestione della coda di stampa, lpq, lprm

Una volta inviato un certo file alla coda di stampa a tale *job* viene assegnato un identificativo numerico progressivo, il *job id*; per vedere l'elenco dei *job* presenti nella coda in attesa di essere stampati, si deve utilizzare il comando `lpq` (*line printer queue*). Per interrompere la stampa di un determinato *job* ed eliminarlo dalla coda di stampa si deve utilizzare il comando `lprm` seguito dal *job id*. Naturalmente la coda di stampa gestisce *job* provenienti da utenti diversi: ogni utente può eliminare dalla coda solo i propri *job*, mentre soltanto *root* può rimuovere dalla coda di stampa lavori inviati da qualunque altro utente.

2.5 Posta elettronica e comunicazione fra utenti

Anche se il nostro sistema non è collegato alla rete Internet o ad una rete locale, in genere dispone del servizio di posta elettronica (*e-mail*) per consentire agli utenti di comunicare fra di loro. Vengono anche resi disponibili alcuni strumenti di comunicazione diretta (*chat*) fra utenti.

Utenti collegati al sistema, whoami, tty, who, finger

Per usare questi strumenti bisogna essere in grado di reperire alcune utili informazioni sugli utenti del sistema; ad esempio, potremmo cominciare da... noi stessi! Il comando `whoami` ci comunica il nostro *username*; visto che è impossibile accedere al sistema senza conoscere il proprio *username*, il comando risulterà più utile quando, trovando un terminale libero, con un utente attivo, ma non presente fisicamente al suo posto di lavoro, vogliamo informarci su chi sia lo sprovveduto che ha dimenticato di effettuare il *logout* prima di andarsene, o anche, richiamato da uno *script* o da un programma *batch*, risulta utile conoscere l'identità dell'utente che sta eseguendo il programma stesso. Il comando `tty` visualizza il nome del terminale (fisico o virtuale) mediante cui siamo connessi al sistema.

Il comando `who` visualizza l'elenco degli utenti collegati in questo momento sul sistema:

```
$ who
root     tty1     Apr 25 12:11
marina   tty5     Apr 25 19:15
marco    ttyp0    Apr 25 18:05
marco    ttyp1    Apr 25 18:32
$ _
```

L'esempio precedente indica che al sistema sono collegati tre utenti, due dei quali (*root* e marina) accedono al sistema da terminali alfanumerici (`tty1` e `tty5`), mentre il terzo (marco) accede al sistema mediante un terminale grafico ed ha aperto due finestre diverse (`ttyp0` e `ttyp1`). Il sistema ci comunica anche l'ora in cui è stato effettuato il *login*.

Un output simile al precedente, ma più dettagliato lo possiamo ottenere con il comando `finger`:

```
$ finger
Login   Name             Tty  Idle  Login Time   Office       Phone
marina  Marina Mori       5   0:02  Apr 25 19:15 [Stanza 418] 2212
marco   Marco Liverani    p0        Apr 25 18:05 [Stanza 203] 2237
marco   Marco Liverani    p0  1:07  Apr 25 18:32 [Stanza 203] 2237
root    Amministratore    1   0:12  Apr 25 12:11
$ _
```

In particolare ci viene comunicato anche il vero nome di ogni utente collegato, eventualmente il nome del suo ufficio ed il numero di telefono; viene anche visualizzato il tempo di inattività (*idle*) dell'utente, cioè da quanto tempo non sta più intergendo con il sistema mediante il mouse o la tastiera.

Lo stesso comando `finger` può essere utilizzato per reperire informazioni ancora più dettagliate su un particolare utente del sistema:

```
$ finger marina
Login: marina                        Name: Marina Mori
Directory: /home/marina              Shell: /bin/bash
Last login Tue Apr 25 19:15 (  ) on tty5
No Mail.
No Plan.
$ _
```

In questo caso, oltre ad alcuni dati già visti, ci viene comunicato anche il nome della *home directory* dell'utente, la *shell* che utilizza e la data dell'ultimo accesso al sistema; il messaggio "`No Mail.`" ci informa che l'utente non ha posta elettronica giacente non ancora letta; se invece ci fossero stati dei nuovi messaggi da leggere, il sistema ci avrebbe informato sulla data e l'ora in cui l'utente ha letto per l'ultima volta la posta. Il messaggio "`No Plan.`" ci informa invece che l'utente non ha predisposto un file per comunicare agli altri delle informazioni sul proprio lavoro o su altri aspetti della propria attività. Se nella *home directory* di Marina si fosse trovato il file "`.plan`" (un normale file di testo), il contenuto di tale file sarebbe stato visualizzato al posto del messaggio "`No Plan.`".

Ultimi collegamenti, last

Per avere un tracciato degli ultimi collegamenti effettuati sul sistema dagli utenti si può utilizzare il comando `last` che, richiamato senza alcun parametro, visualizza la lista (anche molto lunga) degli ultimi collegamenti registrati sul sistema; siccome l'elenco può appunto essere anche molto lungo, conviene richiamare `last` utilizzando `more` per impaginare l'output con il comando "`last | more`". Si può specificare anche l'opzione "$-n$" (n è un qualsiasi numero intero positivo) per visualizzare soltanto gli ultimi n collegamenti; si può pure indicare come parametro lo *username* di un utente per avere la lista dei suoi ultimi collegamenti. Ad esempio:

```
$ last -3
marina  ttyp3    192.168.1.64   Sun Aug 28 19:07   still logged in
marco   ttyp2    woodstock      Sun Aug 28 11:02   still logged in
root    console  localhost      Fri Aug 26 19:54 - 20:17  (00:23)
$ last marco
marco   ttyp2    woodstock      Sun Aug 28 11:02   still logged in
marco   ttyp1    woodstock      Fri Aug 26 19:50 - 20:26  (00:36)
marco   console  localhost      Fri Aug 26 13:09 - 19:00  (05:50)
marco   ttyp3    luna.mat.uniro Thu Aug 25 09:26 - 19:15  (09:48)
...
$ _
```

Ora che sappiamo come verificare se un certo utente è collegato o meno al nostro sistema, possiamo compiere il passo successivo: è possibile fare in modo di visualizzare un messaggio sul suo terminale. Il comando da usare è `write` seguito dallo *username* dell'utente a cui si vuole scrivere ed, eventualmente, anche dal terminale su cui si vuole visualizzare il messaggio; una volta dato il comando si può scrivere il messaggio, anche su più righe; per terminarlo si deve battere Ctrl-d. Immediatamente sul terminale dell'utente specificato verrà visualizzato il messaggio, con tanto di indicazione del mittente.

Inviare un messaggio ad altri utenti, write, wall

Con il comando `wall` (*write all*) è possibile inviare un messaggio sul terminale di tutti gli utenti collegati al sistema; questo comando (che può risultare un po' fastidioso per chi riceve il messaggio), viene utilizzato da *root* per segnalare l'imminente spegnimento del sistema. Anche in questo caso dopo aver dato il comando si può immettere il messaggio da inviare agli utenti, terminando la composizione con i tasti Ctrl-d.

Chat interattiva, talk

Un modo un po' più sofisticato (ed utile) di comunicare con gli altri utenti del sistema è offerto dal comando `talk`, che permette di stabilire una comunicazione bidirezionale con il nostro interlocutore, una specie di telefonata via terminale. La sintassi del comando è come al solito assai semplice: "`talk` *username* [*terminale*]" (come per il `write`, specificare il nome del terminale su cui effettuare il collegamento, non è indispensabile). Se l'utente specificato è effettivamente collegato, il sistema visualizza sul suo terminale un messaggio del tipo:

```
Message from TalkDaemon...
talk: connection requested by marco.
talk: respond with: talk marco
```

Se il nostro interlocutore accetta di iniziare il "talk" dovrà rispondere con un "`talk marco`" e a quel punto la sessione di *chat* avrà inizio: lo schermo dei due terminali viene diviso a metà, nella parte superiore vengono riportate le parole scritte dal proprietario del terminale, mentre in basso sono riportate quelle scritte dal suo interlocutore; per terminare il `talk` si deve battere Ctrl-c.

Questi due tipi di comunicazione (`write` e `talk`) hanno il limite di essere "volatili" quanto una telefonata: terminata la comunicazione di essa non rimarrà traccia da nessuna parte. Richiedono inoltre la presenza contemporanea sul sistema di entrambi gli interlocutori. Viceversa la posta elettronica è uno strumento di maggiore utilità proprio perché non è necessario che il destinatario del messaggio sia collegato al sistema nel momento in cui avviene la spedizione: i messaggi rimarranno giacenti nella *mailbox* dell'utente e potranno essere comodamente letti quando l'utente stesso si collegherà al sistema; in un certo senso è una sorta di potente servizio di segreteria telefonica. Ma non solo: i messaggi di posta elettronica non svaniscono nel nulla dopo che li si è letti, è infatti possibile salvarli su un file o stamparli su carta per poterli rileggere o riutilizzare in seguito. Forse la caratteristica più importante è che i messaggi di *e-mail* possono essere inviati anche a utenti di altri computer, pure se questi non sono utenti di un sistema Unix: infatti il protocollo di trasmissione della posta elettronica utilizzato su Internet e in generale su quasi tutte le reti TCP/IP è un protocollo standard e aperto (SMTP – *Simple Mail Transfer Protocol*) e dunque è stato possibile implementarlo su ogni sistema operativo.

Il processo di trasmissione di una e-mail

In modo estremamente schematico possiamo rappresentare come nel diagramma di Figura 2.1 il processo di spedizione di un messaggio di posta elettronica. Un utente su una postazione *client* (un personal computer o anche una *workstation* Unix), utilizzando un programma di posta elettronica (Eudora, Apple Mail, Mozilla Thunderbird, Microsoft Outlook, ecc.) compone il proprio messaggio di posta indicando l'indirizzo del destinatario e lo invia. Il *client* di posta è configurato per comunicare con un determinato server SMTP che riceve il messaggio e gestisce la spedizione
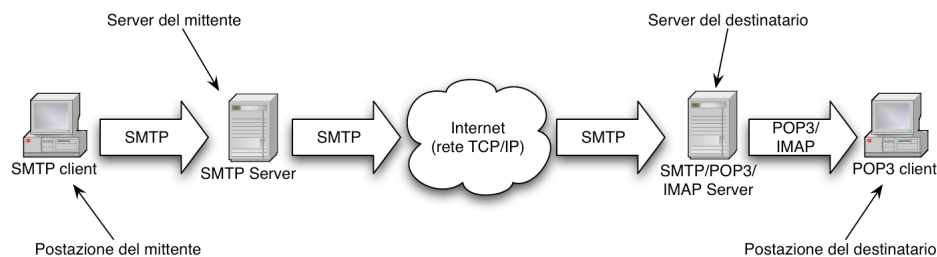
Figura 2.1: Schema del processo di invio/ricezione di un messaggio di e-mail

al destinatario; tipicamente, se il servizio è ben configurato, il server SMTP per la spedizione del messaggio ed il *client* di posta elettronica si trovano sulla stessa rete e comunque il server SMTP "conosce" il *client* o dispone di qualche meccanismo per stabilire che si tratta di una controparte nota e dunque teoricamente fidata. Una volta ricevuto il messaggio il server SMTP lo accoda provando ad inviarlo al server del destinatario ad intervalli di tempo regolari. Non è detto infatti che la consegna del messaggio riesca al primo colpo: ad esempio la rete potrebbe essere difettosa o il server di destinazione potrebbe essere temporaneamente spento o in manutenzione.

Quando finalmente il server di destinazione riceve il messaggio, se riconosce che si tratta di una *mail* destinata ad uno degli utenti di quel server, allora lo accoda nella *mailbox* del destinatario. Quando il destinatario utilizzando un *mail client* decide di verificare se c'è nuova posta in arrivo giacente nella propria *mailbox*, il suo *client* si collega attraverso il protocollo POP3 o IMAP al server di posta e, dopo essersi fatto riconoscere ed autorizzare, effettua finalmente il *download* dei nuovi messaggi.

Frequentemente sulla rete Internet vengono impiegate macchine Unix per realizzare i server di posta elettronica SMTP e POP3, spesso attraverso i programmi `sendmail` (SMTP server) e `popper` o `pop3d` (POP3 server), anche se sta prendendo largamente piede il sistema `qmail`. A meno di non voler utilizzare un client grafico evoluto (come Mozilla Thunderbird, ad esempio) in grado di connettersi ad un server SMTP/POP3 esterno e di sfruttare quindi i servizi offerti dal nostro *Internet Service Provider*, è necessario che la nostra macchina Unix implementi almeno il servizio SMTP per poter inviare messaggi all'esterno.[2] Tuttavia se ci limitiamo allo scambio di messaggi di posta elettronica tra gli utenti di una stessa macchina Unix, allora non è necessario disporre di un server SMTP, dal momento che sono gli stessi *mail client* che consentono di accodare i messaggi nella *mailbox* dei destinatario. Le *mailbox* degli utenti Unix sono infatti, tipicamente, dei semplici file di testo in cui i messaggi di posta elettronica sono riportati uno dopo l'altro. Le *mailbox* sono contenute in file identificati da un nome identico allo *username* dell'utente, memorizzati in `/var/mail` o `/var/spool/mail` (ad esempio il file "`/var/mail/marco`" contiene i messaggi dell'utente `marco`).

Leggere e inviare posta elettronica, mail

Il programma più elmentare per la gestione della posta elettronica è `mail`, presente su quasi ogni sistema Unix, ma se siete un utente alle prime armi forse sarà meglio utilizzare un programma più sofisticato e semplice da usare come `elm` (*Electronic Mail for Unix*) o `pine` (è il diretto concorrente di Elm: PINE=*Program for Internet News & Email*, ma anche *Pine Is Not Elm*). Descriviamo, in estrema sintesi, le principali operazioni da compiere per usare `mail`.

[2]La corretta configurazione di questo servizio è un'operazione complessa e dunque esula dagli obiettivi di questa guida introduttiva; d'altra parte si tratta di una tipica attività di competenza del sistemista Unix che gestisce la nostra macchina, dunque come utenti del sistema possiamo contare sul fatto che tale servizio sia già stato predisposto da qualcun altro.

Per scrivere ed inviare un messaggio di posta elettronica ad un altro utente dovremo semplicemente digitare il comando `mail` seguito dallo *username* dell'utente; il sistema ci chiederà di inserire l'oggetto del messaggio, una sorta di titolo del messaggio stesso, quindi potremo iniziare a digitare il testo. Terminato il messaggio digiteremo all'inizio di una linea vuota un punto, che sta ad indicare la fine del messaggio stesso. Vediamo un esempio:

```
$ mail marina
Subject: cena in pizzeria
Cara Marina,
che ne dici di vederci per cena in pizzeria al Nuovo Mondo
questa sera alle 8?
Ciao,
   Marco
.
EOT
$ _
```

Se Marina è collegata al sistema vedrà comparire sul proprio terminale il messaggio "You have new mail.", altrimenti questo messaggio sarà visualizzato al momento del login, al prossimo collegamento. Per leggere i messaggi giacenti nella *mailbox* Marina non dovrà fare altro che digitare a sua volta il comando `mail`:

```
$ mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/mail/marina": 2 messages 1 new
   1 andrea@woodstock  Sun Aug 28 15:47  21/561  "gita al mare"
>N 2 marco@woodstock   Mon Aug 29 23:21  18/547  "cena in pizzeria"
& 2
Message 2:
From marco@woodstock  Mon Aug 29 23:21:42 2005
To: marina@woodstock
Subject: cena in pizzeria
Date: Mon, 29 Aug 2005 23:21:40 +0200 (CEST)
From: marco@woodstock (Marco Liverani)

Cara Marina,
che ne dici di vederci per cena in pizzeria al Nuovo Mondo
questa sera alle 8?
Ciao,
   Marco

& q
Saved 2 messages in mbox
$ _
```

Se ci sono messaggi giacenti non ancora letti, digitando il comando `mail` si entra in un programma gestito mediante dei comandi formati da una sola lettera che devono essere digitati al *prompt* (che in questo caso è costituito dal carattere "`&`"). Nell'esempio Marina si limita a leggere il messaggio, digitandone il numero progressivo corrispondente, e poi ad uscire dal programma digitando `q` (*quit*). Uscendo dal programma i messaggi giacenti, letti ma non cancellati, vengono archiviati in una *mailbox* "locale" costituita da un file (tipicamente il file `mbox`) collocato nella *home directory* dell'utente, e possono essere recuperati per essere riletti successivamente,

con il comando "`mail -f`" (o "`mail -f mbox`"). Per cancellare un messaggio si deve digitare `d` (*delete*) ed il numero del messaggio; per replicare ad un messaggio si deve digitare `r` (*reply*) ed il numero del messaggio. Battendo `?` si ottiene una lista dei comandi principali del programma `mail`, mentre informazioni più dettagliate le fornisce la pagina di manuale ("`man mail`").

Esiste un altro modo, non interattivo, di spedire messaggi di posta elettronica mediante il programma `mail` e consiste nell'usare le numerose opzioni su linea di comando. La sintassi è la seguente:

`mail -s "`*oggetto*`"` *destinatario* `-c` *altri indirizzi* `<` *file*

dove *oggetto* è l'oggetto del messaggio, racchiuso tra virgolette, *destinatario* è lo *username* dell'utente a cui si intende spedire il messaggio, *altri indirizzi* è una lista di *username* di altri destinatari a cui inviare il messaggio "per conoscenza" (*carbon copy*); l'opzone "`-c`", con gli indirizzi che seguono, può anche essere omessa, visto che è una possibilità in più offerta dal programma, non indispensabile; alla fine della riga si digita il simbolo "`<`" (minore) seguito dal nome del file contenente il testo del messaggio. In questo modo è possibile preparare in precedenza il messaggio con il nostro editor preferito e poi, dopo averlo letto e corretto opportunamente, lo invieremo con questo comando.

Se desideriamo inserire alla fine di ogni nostro messaggio una firma piuttosto elaborata con cui personalizzare le nostre *mail*, è possibile prepararne il testo con un editor e salvarla nella nostra *home directory* con il nome "`.signature`". Il programma di posta elettronica la aggiungerà automaticamente alla fine di ogni messaggio in partenza.

Altri programmi di posta elettronica, Pine, Elm

Lavorare con il programma `mail` è comunque abbastanza laborioso e di certo poco intuitivo; molto più semplice e guidato, come abbiamo già accennato, è invece l'uso di `elm` o del suo diretto concorrente `pine`. Quest'ultimo in particolare è guidato completamente mediante dei menù molto chiari e dispone di un editor (`pico`) estremamente comodo da usare (vedi Figura 2.2); inoltre arricchire la mail con dei file allegati (*attachment*) è un'operazione che con `pine` è molto semplice.

2.6 Gestione dei processi

In ambiente Unix si parla di *processo* per indicare un programma in esecuzione. Sappiamo che Unix è un sistema operativo *multitasking*, ma fino ad ora abbiamo sfruttato questa caratteristica solo grazie alla multiutenza, che ci permetteva di "toccare con mano" il fatto che la macchina, avendo più utenti collegati contemporaneamente, stava effettivamente elaborando più di un programma.

Per sfruttare pienamente e con comodità il *multitasking* si deve disporre di un terminale grafico che ci permetta di aprire sullo schermo più finestre in cui lanciare contemporaneamente diversi processi in modalità interattiva; è possibile fare altrettanto su un normale terminale alfanumerico, ma anche in questo caso si deve disporre di un programma (ad esempio `screen`) che ci consenta di simulare un ambiente con più finestre. Vedremo in maggiore dettaglio l'ambiente X Window in seguito, per ora ci limiteremo a dire che è possibile lanciare delle applicazioni che lavorano autonomamente ed indipendentemente dalle altre in una regione dello schermo (finestra) riservata ad ognuna di esse.

Per lanciare un'applicazione, come al solito è necessario dare un comando al *prompt* della *shell*, ma così facendo, come abbiamo visto fino ad ora, non possiamo utilizzare contemporaneamente sullo stesso terminale un altro programma, almeno fino a quando non saremo usciti dal programma in esecuzione. Esiste un modo per sganciare il processo "figlio" (il programma da eseguire) dal processo "padre" (ad

```
  PINE 4.63   MAIN MENU                          Folder: INBOX  2 Messages

        ?     HELP                -  Get help using Pine

        C     COMPOSE MESSAGE     -  Compose and send a message

        I     MESSAGE INDEX       -  View messages in current folder

        L     FOLDER LIST         -  Select a folder to view

        A     ADDRESS BOOK        -  Update address book

        S     SETUP               -  Configure Pine Options

        Q     QUIT                -  Leave the Pine program

Copyright 1989-2005.  PINE is a trademark of the University of Washington.

? Help                    P PrevCmd               R RelNotes
O OTHER CMDS > [ListFldrs] N NextCmd               K KBLock
```

Figura 2.2: Il menù principale del programma Pine

esempio la *shell* da cui si lancia il programma) rendendo i due processi indipendenti ed autonomi. Se aggiungiamo il carattere "&" (e commerciale) alla fine della riga che invoca un certo comando, tale comando sarà eseguito in una "sessione" separata e quindi potremo utilizzare contemporaneamente la *shell* (da cui possiamo nel frattempo lanciare altri processi) ed il programma che abbiamo lanciato. Questo è possibile a patto che il programma "figlio" non richieda una interazione da parte dell'utente mediante il terminale (ad esempio un input da tastiera, o una visualizzazione molto "verbosa" di messaggi in output).

Interruzione e sospensione di un programma, Control-c, Control-z

Vediamo un esempio molto semplice che possiamo provare anche su un normale terminale alfanumerico. Il comando `yes` visualizza una serie di "y" sullo schermo fino a quando non viene interrotto dall'utente. Per interrompere l'esecuzione di un programma dobbiamo battere Ctrl-c (*break*). Se invece di interromperlo volessimo solo sospenderne temporaneamente l'esecuzione, invece di *break* dovremmo battere Ctrl-z (*stop*):

```
$ yes
y
y
y
...
(l'utente batte Ctrl-z)
[1]+  Stopped           yes
$ _
```

Elenco dei processi attivi in una shell, jobs

Il sistema ci informa che il programma identificato dal *job number* 1, che è stato lanciato con il comando `yes`, è stato momentaneamente interrotto. Il comando `jobs` ci permette di visualizzare la lista dei processi lanciati da quella *shell*; nel caso dell'esempio precedente avremmo il seguente output:

```
$ jobs
[1]+  Stopped           yes
```

```
$ _
```

In questo momento abbiamo due processi attivi: la *shell* ed il programma `yes` che è momentaneamente interrotto. In particolare diremo che il processo attivo, la *shell*, è in *foreground*, mentre l'altro processo è in *background*; in uno stesso momento, nella sessione attiva su un determinato terminale, può esserci un solo programma in foreground, ma anche molti programmi in background. Per riportare in foreground il programma `yes` (e mandare quindi in background la *shell*) si deve usare il comando `fg` (*foreground*) seguito dal numero del *job*:

Processi in foreground e in background

```
$ fg 1
y
y
y
...
```

Avevamo citato precedentemente tra le unità disponibili su un sistema Unix, anche il device "nullo" `/dev/null`. Ora potrebbe tornarci utile. È possibile redirigere l'output di una applicazione verso una unità diversa dal *device* di output standard (il video del terminale) mediante l'uso del carattere "`>`" (maggiore). Proviamo a redirigere l'output del programma `yes` verso l'unità nulla, dando il comando "`yes > /dev/null`". Il programma è in esecuzione, ma sullo schermo non appare nulla, neanche il *prompt* della *shell* per poter lanciare altri programmi nel frattempo. Interrompiamo l'esecuzione di `yes` battendo Ctrl-c e proviamo a riavviarlo con il comando "`yes > /dev/null &`":

```
$ yes > /dev/null &
[1] 143
$ jobs
[1] 143  Running          yes >/dev/null &
$ ps
  PID TTY STAT  TIME COMMAND
   67  1  S     1:32 bash
  143  1  R     0:02 yes
  152  1  R     0:00 ps
$ _
```

Con l'aggiunta del simbolo "`&`" alla fine della linea di comando, abbiamo lanciato l'applicazione in *background*, mantenendo l'uso della *shell* per impostare altri comandi. Con il messaggio "`[1] 143`" il sistema ci comunica che l'applicazione `yes` è il primo processo lanciato da questa *shell* e, nella tabella di tutti i processi del sistema, gli è stato assegnato il numero 143 (questo non vuol dire che ci sono 143 processi attivi). Con il comando `jobs` verifichiamo gli stessi dati ed in più il sistema ci comunica che l'applicazione è attualmente in esecuzione (*running*). Il comando `ps` ci fornisce delle informazioni su tutti i nostri processi attivi, non solo quelli lanciati attraverso una certa *shell*; l'esempio mostra che nel nostro caso sono attivi tre processi, tutti sul terminale `tty1`: `bash`, la *shell* è attiva da un'ora e 32 minuti ed è momentaneamente sospesa ("`S`") perchè sta eseguendo il comando `ps`, che è stato appena attivato ed è in esecuzione ("`R`"), come pure il programma `yes`, attivo anche questo solo da qualche istante. Osserviamo infine che con il comando `bg` è possibile riavviare in *background* un processo precedentemente sospeso con Ctrl-z. Il comando `bg` accetta come argomento il *job id* preceduto dal simbolo di percentuale o il numero del processo.

Lista dei job e dei processi attivi, jobs, ps

Come abbiamo già accennato nelle pagine precedenti ogni programma attivo su un sistema Unix viene eseguito a nome e per conto di un determinato utente e da questo eredita i permessi per le operazioni sul *filesystem*; i processi di sistema vengono eseguiti, in genere, a nome dell'utente *root*: anche se questo non li ha lanciati digitando un comando sul terminale, durante la fase di *bootstrap* (la sequenza di *start-up* del sistema eseguita subito dopo l'accensione o un riavvio della macchina) vengono lanciati numerosi processi che rimarranno attivi (anche se in *background*) fino al successivo *shutdown* del sistema. Questi processi vengono chiamati in gergo "demoni" (*daemons*). Con il comando `ps` è possibile visualizzare anche l'elenco di tutti i processi attivi sul sistema; nelle opzioni del comando necessarie per compiere tale operazione, emerge una delle poche differenze tra Unix di tipo BSD e System V visibili anche all'utente generico. Sui sistemi BSD per visualizzare tutti i processi attivi bisogna aggiungere al comando `ps` le opzioni "`-auxw`", mentre sugli Unix SVR4 le opzioni sono "`-ef`"; l'output è molto simile in entrambi i casi; di seguito riportiamo un esempio estratto dall'output prodotto su una macchina Sun Solaris 8 (dunque una versione di Unix System V):

```
$ ps -ef
   UID   PID  PPID  C    STIME TTY      TIME CMD
  root     0     0  0   Aug 16 ?        0:16 sched
  root     1     0  0   Aug 16 ?        0:52 /etc/init -
  root     2     0  0   Aug 16 ?        0:00 pageout
  root     3     0  0   Aug 16 ?       67:57 fsflush
   www   297   292  0   Aug 16 ?        0:38 /usr/apache/bin/httpd
  root   112     1  0   Aug 16 ?        0:00 /usr/sbin/in.routed -q
  root   284     1  0   Aug 16 ?        0:00 /usr/lib/nfs/mountd
  root   255     1  0   Aug 16 ?        0:31 /usr/local/sbin/sshd
  root   155     1  0   Aug 16 ?        1:01 /usr/sbin/inetd -s
  root   172     1  0   Aug 16 ?       16:26 /usr/sbin/syslogd
   www   299   292  0   Aug 16 ?        0:39 /usr/apache/bin/httpd
 marco 10216 10214  0 21:50:29 pts/1    0:00 -tcsh
  root   292     1  0   Aug 16 ?        0:04 /usr/apache/bin/httpd
...
```

L'output viene incolonnato e sulla prima riga sono riportate delle etichette che aiutano ad interpretare il significato delle righe successive: sulla prima colonna viene riportato lo *username* (UID) dell'utente che sta eseguendo il processo; nella seconda colonna viene visualizzato il *process id* (PID), ossia il numero progressivo che identifica univocamente un determinato processo; subito dopo, nella terza colonna, è riportato il numero identificativo del processo padre (PPID, *parent process id*), ossia del processo che ha eseguito il processo a cui si riferisce la riga in esame. Come si vede nell'esempio il programma `sched` è il primo ad essere lanciato (il PID è 0) e da quel processo sono stati lanciati altri tre processi: `/etc/init` (PID=1), `pageout` (PID=2) e `fsflush` (PID=3). A sua volta il programma `init` ha lanciato molti altri processi. È interessante notare che il padre del processo con ID=0 è il processo stesso (PID=PPID=0): deve pur esistere un processo da cui ha origine lo *start-up* del sistema operativo e dei processi principali, tuttavia, interpretando i codici PID e PPID di tale processo, sembra proprio che sia nato da se stesso; questo giustifica l'espressione gergale *bootstrap* che richiama l'idea che la macchina per partire (o per "salire", altra espressione nel gergo degli utenti Unix) sembrerebbe essersi sollevata da terra tirandosi per i lacci delle scarpe!

Gerarchia dei processi

Nella colonna TTY viene indicato il terminale da cui è stato lanciato il processo; come si vede nell'esempio molti processi non sono associati a nessun terminale, visto che sono demoni lanciati automaticamente durante la fase di *boot* del sistema.

Con il comando `kill` è possibile interrompere forzatamente l'esecuzione di un processo (come la pressione dei tasti Ctrl-c per i processi in *foreground*). Insieme a `kill` si deve specificare come parametro il PID dell'applicazione che vogliamo terminare o il suo *job number*, preceduto però dal simbolo di percentuale "`%`". Ad esempio per interrompere il programma `yes` dell'esempio di pagina 25 i due comandi "`kill 143`" e "`kill %1`" sono equivalenti.

Terminare un processo, kill

Il comando `kill` accetta alcune opzioni che consentono di modificare il segnale inviato al processo che si intende terminare: "`-TERM`" è l'opzione di default e invita il processo a terminare normalmente; l'opzione "`-KILL`" è invece più brutale e termina il processo immediatamente, anche se questo sembra non reagire a comandi di terminazione più "morbidi"; l'opzione "`-HUP`" infine consente, per quei processi che lo prevedono, di terminare e rilanciare il processo stesso.

2.7 Variabili di ambiente e alias di comandi

Come abbiamo già accennato nelle pagine precedenti la *shell* Unix consente di definire delle variabili di sessione e di ambiente in cui è possibile memorizzare delle informazioni; tipicamente vengono usate per memorizzare durante la sessione di lavoro (le variabili vengono infatti cancellate automaticamente quando la sessione termina) alcune impostazioni e parametri di configurazione che consentono di rendere più confortevole la sessione di lavoro stessa. Le variabili di sessione valgono solo nell'ambito della *shell* in cui sono state definite; viceversa è possibile definire delle variabili di ambiente la cui visibilità è estesa anche a tutti i programmi che vengono lanciati dalla *shell* in cui la variabile è stata definita.

Per definire una variabile di sessione impostando il nome ed un valore per la variabile stessa si deve usare una sintassi differente a seconda della *shell* che si sta utilizzando. Sotto `bash` si deve semplicemente digitare "*nome*=*valore*" (ad esempio "`pippo=pluto`"); sotto `tcsh` invece si deve usare il comando `set` con la sintassi "`set` *nome*=*valore*" (ad esempio "`set pippo=pluto`").

Definizione di variabili di sessione e di ambiente, set, setenv, export

Anche per la definizione di variabili di ambiente si devono usare comandi differenti a seconda della *shell* utilizzata: su `bash` il comando è `export`, mentre su `tcsh` il comando equivalente è `setenv`. Ad esempio si potrà digitare il comando "`export pippo=pluto`" oppure "`setenv pippo pluto`" per definire la variabile di ambiente `pippo` con il valore `pluto`.

In tutti i casi per eliminare una variabile si può usare il comando `unset` con la sola eccezione delle variabili di ambiente sotto `tcsh` che devono essere eliminate con il comando `unsetenv`.

Per visualizzare l'elenco delle variabili di ambiente definite con qualche valore si usa il comando `set` senza alcun parametro. Per visualizzare il valore di una variabile specifica si può invece usare il comando `echo`; per distinguere le stringhe di testo dai nomi delle variabili, questi ultimi devono essere preceduti dal carattere "`$`":

```
$ pippo=pluto
$ echo $pippo
pluto
$ echo pippo
pippo
$ unset pippo
$ _
```

Una variabile di ambiente molto importante è la variabile `PATH`: quando diamo un comando, la *shell* va a cercare il file eseguibile con quel nome solo nelle directory del *filesystem* specificate nella variabile di ambiente `PATH`. Questa variabile viene

Il path

usata ad esempio dal comando `which` che ci permette di scoprire in quale delle directory del *path* è collocato fisicamente il file eseguibile di un certo programma (ad esempio: "`which ls`"). La variabile `PATH` contiene una sequenza di directory separate tra loro dal carattere "`:`"; per aggiungere una certa directory (ad esempio `/home/marco/bin`) al *path* si può usare il seguente comando sotto Bash (o un comando equivalente sotto C Shell):

```
$ export PATH=$PATH:/home/marco/bin
$ echo $PATH
/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/home/marco/bin
$ _
```

Definizione del prompt della shell

Un'altra variabile di ambiente assai utile è quella che consente di definire il *prompt* della linea di comando. Anche in questo caso la sintassi dipende dalla *shell* che stiamo utilizzando: sotto `bash` la variabile da impostare si chiama `PS1`, mentre sotto `tcsh` si chiama `prompt`. Per definire la stringa da utilizzare come *prompt* si possono usare delle sequenze di caratteri che hanno un significato specifico. Ad esempio spesso si utilizza un *prompt* che indica lo *username* dell'utente, l'hostname della macchina su cui sta lavorando ed il *path* della directory corrente (ad esempio "`marco@woodstock ~/src/perl$`"); per ottenere questo *prompt* si deve dare il seguente comando su `bash` "`export PS1=\u@\h \w\$`", dove "`\u`" rappresenta il nome dell'utente, "`\h`" l'hostname della macchina e "`\w`" il *path* della directory corrente. Su `tcsh` invece il comando da utilizzare per ottenere lo stesso risultato è "`setenv prompt "%n@%m %c3%#"`".

Alias di comandi, alias

Visto che la maggior parte dei comandi Unix ammette numerosissime opzioni, alcune delle quali sono effettivamente di uso molto comune, la *shell* consente di definire degli *alias* con cui rinominare i comandi complessi. Il comando `alias` consente di assegnare un nome ad un'istruzione anche molto estesa; ad esempio:

```
$ alias nproc='ps -auxw|wc -l'
$ alias ls='ls -F'
$ alias rm='rm -i'
$ _
```

Il comando `alias` senza alcun parametro visualizza l'elenco degli alias definiti nella sessione.

2.8 Altri comandi utili

I comandi disponibili su un sistema Unix sono alcune centinaia: oltre ad essere impossibile riportarli tutti, sarebbe anche abbastanza inutile, visto che sono moltissimi i comandi che non ho mai usato e che probabilmente non userò mai. Ci limitiamo in questa sezione a descrivere per sommi capi alcuni comandi non indispensabili, ma che possono essere di una certa utilità. Per quanto riguarda gli altri, quelli che non hanno trovato spazio in queste pagine, il consiglio è il seguente: quando sentirete la necessità di un comando per svolgere un particolare compito, cercatelo, magari "sfogliando" le pagine di manuale, perché quasi sicuramente già esiste!

Redirezione dell'input e dell'output

Abbiamo spesso usato, negli esempi della pagine precedenti, gli operatori di *redirezione* dell'input/output. Più in dettaglio possiamo dire che il simbolo "`>`" serve per inviare l'output standard (quello che normalmente finisce sul video del terminale) su un file o su un altro *device*; la sequenza "`>>`" consente di accodare l'output ad un file già esistente, senza cancellarne il contenuto. Per redirigere i messaggi di errore prodotti da un programma, che generalmente non sono inviati su *standard output*, ma su un canale denominato *standard error*, si deve usare l'operatore

"2>". Il simbolo "<" serve invece per leggere l'input standard (quello che altrimenti sarebbe inserito manualmente dall'utente mediante la tastiera) da un file o da un altro *device*. Infine il simbolo "|" (*pipe*) serve a collegare il canale di output di un programma verso il canale di input di un altro programma: in questo modo ciò che verrebbe normalmente visualizzato in output eseguendo il primo programma, viene invece utilizzato come input per il secondo. Il seguente esempio utilizza tutti gli operatori di redirezione dell'I/O:

```
$ cat < lista | sort > lista.ordinata
$ _
```

Il programma `cat` (che, come abbiamo visto, serve a visualizzare il contenuto di un file) riceve l'input dal file `lista`; l'output di `cat` viene inviato mediante il *pipe* al programma `sort` (che serve ad ordinare i dati contenuti in una lista) che li invia in output al file `lista.ordinata`. Al termine dell'esecuzione di questo comando il file `lista.ordinata` conterrà gli stessi dati di `lista`, ma ordinati alfabeticamente.

Ordinamento di un file, sort

Riguardo al programma `sort` è forse opportuno aggiungere che i comandi "`sort < lista > lista.ordinata`" e "`sort lista -o lista.ordinata`" avrebbero svolto efficacemente lo stesso compito del comando riportato nell'esempio precedente al solo scopo di illustrare l'uso di tutti i simboli di redirezione dell'I/O. Le opzioni "`-r`" e "`-n`" producono rispettivamente l'ordinamento inverso (*reverse*) e l'ordinamento ottenuto considerando le righe del file in input come numeri (per cui "`10`" è maggiore di "`9`", e non il contrario come avverrebbe con l'ordinamento alfabetico). Concatenando opportunamente il comando `sort` con `du` e `head` si può ottenere la "classifica" dei file che occupano più spazio in una certa directory, come nel seguente esempio che permette di visualizzare le tre *mailbox* più capienti tra quelle presenti nella directory `/var/mail`:

```
$ du -sk /var/mail/* | sort -rn | head -3
199904  /var/mail/root
54712   /var/mail/andrea
47600   /var/mail/marco
$ _
```

Backtick

Esiste un altro modo per passare ad un comando o ad un programma l'output prodotto da un altro processo: il cosiddetto *backtick*. Consiste nel richiamare un determinato comando racchiudendolo tra "apici inversi" (il carattere con codice ASCII 96, da non confondere con l'apostrofo); in questo modo viene prodotta una stringa con l'output del comando, che può essere passata come parametro ad un altro programma. La differenza rispetto al *pipe* è sostanziale: con il *backtick* l'output di un processo non viene passato in input, come con il *pipe*, ma può essere utilizzato come parametro di un altro programma. Vediamo il seguente esempio in cui sono usati i comandi `which` e `file` che abbiamo già descritto nelle pagine precedenti:

```
$ which ls
/bin/ls
$ file `which ls`
/bin/ls: ELF 32-bit MSB executable SPARC Version 1, dynamically
linked, stripped
$ _
```

Ricerca di stringhe, espressioni regolari, grep

Uno strumento molto importante ed utile per operare sui file di testo è il comando `grep`: consente di cercare in un insieme di file una sequenza di caratteri (in gergo una *stringa*) rappresentata da un *pattern* descritto con una espressione regolare.

Un'espressione regolare è una sequenza di caratteri che esprime un pattern per identificare delle stringhe, utilizzando una sintassi estremamente potente in una forma sintetica e compatta. Ci limiteremo a descrivere solo poche regole per la composizione delle espressioni regolari, rimandando alla documentazione di sistema del comando `grep` per una illustrazione approfondita di questo potentissimo strumento. In una espressione regolare ogni carattere rappresenta se stesso, con poche eccezioni costituite da caratteri che assumono un significato speciale. Tra questi il punto "`.`" indica un carattere qualsiasi, mentre i simboli "`?`", "`+`" e "`*`" sono dei quantificatori: rispettivamente indicano che il carattere o l'espressione che li precede deve comparire nella stringa zero o una volta, una o più volte, zero o più volte. Per indicare un carattere che nella sintassi delle espressioni regolari assume un significato speciale, basta farlo precedere dal carattere "\" (*backslash*); dunque la sequenza "\\" in una espressione regolare rappresenta proprio il carattere *backslash*.

L'espressione regolare più generica è costituita dalla sequenza "`.*`", che rappresenta una stringa qualunque di qualsiasi lunghezza (compresa la stringa vuota); l'espressione regolare "`ab.*b.a`" rappresenta tutte le stringhe che iniziano con "`ab`", poi proseguono con una sequenza anche nulla di caratteri qualsiasi, quindi contengono il carattere "`b`", poi un carattere qualunque ed infine il carattere "`a`". Corrispondono a questa espressione regolare le stringhe "`abbca`", "`abxyzwbza`" e infinite altre; l'espressione regolare "`a.*.txt`" rappresenta tutte le stringhe che iniziano con la lettera "`a`", sono seguite almeno da un carattere e terminano con "`txt`".

Dunque con `grep` possiamo visualizzare tutte le righe di un file che contengono una sottostringa che corrisponde con il pattern rappresentato dall'espressione regolare riportata come parametro e delimitata da apici. L'opzione "`-v`" consente di invertire la selezione: tutte le righe del file che non contengono la stringa rappresentata dall'espressione regolare. L'opzione "`-i`" consente di effettuare una ricerca *case insensitive*, ossia considerando come uguali fra loro le lettere maiuscole e minuscole.

```
$ grep -i 'rossi' indirizzi.txt
Mario Rossi, via marmorata, Roma, 06 9182736
$ grep -v 'Roma' indirizzi.txt
Lorenzo Bianchi, via monte bianco, Milano, 02 7349014
Giuliano Verdi, v.le beethoven, Firenze, 055 8123492
Manuela Bianchi, via monte bianco, Milano, 02 7349014
$ _
```

Il comando `grep` è molto utile anche come filtro dell'output di un altro programma particolarmente "verboso"; il seguente comando, ad esempio, visualizza l'output del comando `ps` escludendo tutte le righe che contengono la stringa "`root`":

```
$ ps -ef | grep -v 'root'
UID         PID   PPID   C STIME    TTY      TIME CMD
daemon      795      1 0.0   Aug 22 ??    0:23.57 lpd Waiting
smtp      26927      1 0.0   Aug 31 ??    0:00.07 /usr/lib/sendmail
andrea   172264      1 0.0 17:19:28 ??    0:00.55 mwm -multiscreen
marco    198161 198152 0.0 22:43:40 pts/1 0:00.08 -tcsh (tcsh)
luca     184062 184244 0.0 19:30:41 pts/2 0:00.29 pine
luca     184244 184230 0.0 19:30:38 pts/2 0:00.07 -tcsh (tcsh)
$ ps -ef | grep -v 'root' | wc -l
7
$ _
```

Data e ora, calendario perpetuo, date, cal

Con il comando `date` il sistema ci fornisce la data e l'ora corrente; l'amministratore di sistema (*root*) con lo stesso comando può anche impostare la data e

l'ora. Il comando `cal` senza alcun parametro ulteriore visualizza in forma sintetica il calendario del mese corrente; per avere il calendario completo di un intero anno basterà specificare l'anno desiderato di seguito al comando `cal`; così ad esempio "`cal 1789`" ci permette di scoprire che il 14 luglio 1789, giorno della presa della Bastiglia, era un martedì. Specificando due parametri il primo verrà interpretato come il numero progressivo di un mese ed il secondo come il numero dell'anno di cui si intende visualizzare il calendario; ad esempio:

```
$ cal 7 1789
     July 1789
 S  M Tu  W Th  F  S
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31
$ _
```

Promemoria degli appuntamenti, calendar

Il programma `calendar` è un programma di utilità che consente di avere un promemoria delle attività da svolgere nei prossimi giorni. È basato sul contenuto di un file di testo denominato `calendar` che deve essere presente nella *home directory* dell'utente; il file deve contenere una attività (o un appuntamento) per ogni riga e le righe del file devono iniziare con una data nel formato "*mm/gg*" (ad esempio "6/7" per il 7 giugno). Ogni volta che il comando `calendar` viene invocato questo produce la stampa delle righe relative alla data odierna e al giorno successivo; il venerdì stampa gli appuntamenti del giorno stesso e dell'intero week-end. Vediamo un esempio:

```
$ date +%d/%m/%Y
01/09/2005
$ tail -6 calendar
08/30 Telefonare ad Aurora per prossimo incontro
08/30 Passare in biblioteca (chiude alle 13!)
09/01 Consegnare dispense del corso di crittografia
09/01 Riunione con Paola alle 11:00
09/02 Ricevimento studenti (9-13)
09/03 In pizzeria con Marina, Rita e Andrea
$ calendar
09/01 Consegnare dispense del corso di crittografia
09/01 Riunione con Paola alle 11:00
09/02 Ricevimento studenti (9-13)
$ calendar | mail liverani@mat.uniroma3.it
$ _
```

Compressione di file, gzip, compress, zip, tar

A volte si desidera tenere un file sul disco, magari senza utilizzarlo spesso; in tal caso, per risparmiare spazio, potrebbe essere opportuno comprimere tale file, ricodificandolo in modo opportuno. A questo scopo esistono i programmi di compressione che si occupano di generare un file in formato compresso che, pur contenendo le stesse informazioni, occupa meno spazio del file originale. Naturalmente quando vorremo accedere alle informazioni contenute nel file originale, dovremo eseguire l'operazione inversa, scompattando il file compresso, per ottenere nuovamente il file originale. Su Unix esistono implementazioni di numerosi programmi compattatori: `gzip` e `compress` sono sicuramente i più diffusi in questo ambiente. Con il comando "`gzip` *file*" si comprime il file eliminando il file originale ed ottenendo al suo posto

un file con estensione "`.gz`"; per ottenere il file originale da quello compresso si deve usare il comando inverso "`gunzip` *file.gz*" (o anche "`gzip -d` *nome.gz*"). Sintassi del tutto analoga è quella del comando `compress` (e del suo inverso `uncompress`). I file compressi con `gzip` hanno estensione "`.gz`", mentre quelli trattati con `compress` hanno estensione "`.Z`".

Un po' diverso è il programma `zip`, che produce file compressi nello stesso formato utilizzato dai programmi `PKZIP` e `WinZip`, molto diffusi sui sistemi Windows. È possibile archiviare numerosi file all'interno di un unico file compresso. Ad esempio con il comando "`zip sorgenti.zip src/*`" si archiviano in formato compresso, nel file `sorgenti.zip`, tutti i file contenuti nella directory `src`. Per estrarre i file originali dal file "zippato" si deve usare il comando "`unzip sorgenti.zip`", mentre per vedere il contenuto del file compresso, senza però estrarre i file in esso contenuti, si può usare l'opzione "`-v`"; ad esempio "`unzip -v src.zip`".

Archiviazione di file, tar

Il comando `tar` (*tape archive*) è molto usato in ambiente Unix e di solito serve per archiviare (senza comprimere) in un file unico più file, magari sparsi in directory differenti. In particolare è molto usato per effettuare il *backup* dei file del sistema su un nastro. I file archiviati con `tar` hanno estensione "`.tar`", mentre quelli con estensione "`.tgz`" sono file in formato `tar` compressi con `gzip`. Per archiviare un insieme di file (e directory) in un file in formato "tar" si deve usare l'opzione "`cfv`": `c`=*create*, `f`=*file* (opera su file e non su nastro), `v`=*verbose* (visualizza sullo schermo del terminale i nomi dei file archiviati); ad esempio, per archiviare in "`archivio.tar`" tutto il contenuto della directory corrente si può usare il comando "`tar cfv archivio.tar .`". Per visualizzare il contenuto di un file archiviato con `tar` si devono usare le opzioni "`tfv`" (ad esempio "`tar tfv archivio.tar`"), mentre per estrarre i file si devono usare le opzioni "`xfv`" (ad esempio "`tar xfv archivio.tar`").

```
$ ls *.txt
lettera.txt  nota_tecnica.txt       pippo.txt
$ tar cfv archivio.tar *.txt
a lettera.txt 2K
a nota_tecnica.txt 5K
a pippo.txt 1K
$ tar tfv archivio.tar
tar: blocksize = 8
-rw------- 107/60001     2159 Feb  6 16:50 2005 lettera.txt
-rw-rw-r-- 107/60001     5079 Nov 30 17:53 2000 nota_tecnica.txt
-rw-r--r-- 107/60001      762 Nov 30 17:55 2000 pippo.txt
$ tar xfv archivio.tar pippo.txt
tar: blocksize = 8
x pippo.txt, 762 bytes, 1 tape blocks
$ _
```

Calcolatrice, bc

Sulla linea di comando della *shell* è possibile attivare una potente calcolatrice con il comando `bc`. Si tratta di un ambiente di calcolo molto più potente di una semplice calcolatrice da tavolo, con cui è possibile definire anche variabili e funzioni, ma che, usato in modo più elementare, consente di eseguire facilmente dei calcoli. Per uscire dall'ambiente `bc` si può digitare il comando `quit` o battere i tasti Ctrl-d. L'opzione "`-l`" impone l'utilizzo della libreria matematica standard, per cui il risultato dei calcoli sarà visualizzato utilizzando anche la parte decimale di valori non interi; senza questa opzione i risultati delle operazioni vengono visualizzati troncando la parte decimale.

```
$ bc -l
bc 1.06
```

```
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type ‘warranty’.
(3+2)/2
2.50000000000000000000
quit
$ echo "(3+2)/2" | bc -l
2.50000000000000000000
$ _
```

Il programma `bc` può essere usato anche in modalità non interattiva, passando le espressioni da calcolare attraverso un file specificato come argomento sulla linea di comando o dal canale di input standard (*standard input*) tramite l'operatore di *pipe*, come nell'esempio precedente.

2.9 Le pagine di manuale

La fonte di informazione principale sui comandi Unix e sui programmi installati sul proprio sistema è costituita dalle cosiddette *man pages*, o pagine di manuale, che formano, in unione ad un comodo programma di consultazione, una vera e propria biblioteca di manuali *on-line*, sempre pronti ad essere consultati. L'insieme delle man pages è suddiviso per argomento in nove sezioni:

Sezioni delle pagine del manuale Unix

1. User command, dove sono riportati tutti i comandi utili per l'utente;
2. System calls, vengono descritte tutte le funzioni standard del linguaggio C per effettuare delle chiamate alle funzioni del sistema (utile soprattutto ai programmatori);
3. Subroutines, vengono descritte le *subroutine* e le funzioni del sistema di sviluppo (linguaggio C);
4. Devices, dove sono riportate le descrizioni dettagliate dei devices installati sul sistema;
5. File Formats, vengono riportati i formati dei principali file di configurazione del sistema;
6. Games, descrizione dei giochi installati sul sistema;
7. Miscellaneous, altre descrizioni che non trovano collocazione migliore nelle altre sezioni;
8. System administration, descrizione dei comandi per l'amministratore del sistema (*root*);

n. New, nuove pagine di manuale ancora non inserite nelle rispettive sezioni;

Ogni pagina di manuale è contenuta in un file diverso ed i file appartenenti ad una stessa sezione sono contenuti nella stessa directory; ad esempio i file con le descrizioni dei comandi della prima sezione possono trovarsi nella directory `/usr/man/man1`.

Se è presente nel sistema la pagina di manuale relativa ad un certo programma, potremo visualizzarla mediante il comando `man` seguito dal nome del programma. La visualizzazione mediante il comando `man` viene filtrata automaticamente attraverso `more` e quindi potremo scorrere facilmente il testo della pagina, anche se questa dovesse risultare molto lunga. Il comando `apropos` (equivalente a "`man -k`") con-

Visualizzazione del manuale, man, xman, apropos

sente di cercare fra le pagine del manuale quelle che riguardano il termine specificato come argomento.

Se invece di operare mediante un terminale alfanumerico stessimo lavorando su un X Terminal, potremmo usare, invece di `man`, il comando `xman` che ci permette di "sfogliare" le pagine del manuale mediante una interfaccia a menù utilizzabile con il mouse.

Ad esempio con "`man man`" si visualizzano le istruzioni per l'uso dello stesso manuale, mentre con "`man mkdir`" si visualizza la pagina di manuale relativa al comando `mkdir`:

```
$ man mkdir
MKDIR(1L)                                         MKDIR(1L)

NAME
      mkdir - make directories

SYNOPSIS
      mkdir [-p] [-m mode] [--parents] [-mode=mode] [--help]
      [--version] dir...

DESCRIPTION
      This manual page  documents  the GNU version of mkdir.
      mkdir  creates a directory  with  each  given name. By
      default, the mode of created directory is  0777  minus
      the bits set in the umask.

   OPTIONS
      -m, --mode mode
             Set the mode of created  directories  to  mode,
...
$
```

Le *man pages* hanno tutte un formato piuttosto simile: innanzi tutto viene descritta in modo sintetico la sintassi del comando (*synopsis*) quindi vengono elencate e descritte dettagliatamente le opzioni che è possibile specificare insieme al comando. Di solito alla fine della pagina è riportato l'elenco dei file di configurazione del programma stesso (*files*), l'elenco di altri comandi correlati (*see also*) ed eventuali problemi noti, riscontrati in situazioni particolari nell'uso del programma (*bugs*).

Figura 2.3: Terminale alfanumerico Digital VT100

Capitolo 3

Editing di file di testo

In questo capitolo descriveremo sinteticamente tre programmi per comporre file di testo. Ulteriori e più approfondite informazioni su questi editor possono essere reperite, come al solito, sulle relative *man pages*. I primi due editor sono tra i più diffusi in ambiente Unix: `vi` ed `emacs`; il terzo è `pico`, l'editor del programma di posta elettronica `pine`, che può essere usato anche indipendentemente da quest'ultimo; `pico`, pur essendo meno diffuso dei primi due e soprattutto molto meno potente, è stato inserito in questa brevissima rassegna perché è uno degli editor più semplici da usare tra i tanti reperibili in ambiente Unix.

La variabile di ambiente `EDITOR` contiene il nome del programma editor di default, che viene invocato automaticamente in determinate circostanze. Se la variabile non esiste (non è stata definita) allora l'editor di default è `vi`. Per cambiare l'editor di default si deve modificare il valore di questa variabile.

3.1 L'editor vi

`vi` (si pronuncia "*vuai*") è sicuramente l'editor più diffuso sotto Unix ed anche uno degli editor più potenti in assoluto. Con `vi` si può fare praticamente tutto, l'unico difetto è che è un po' ostico da utilizzare per chi è alle prime armi ed è completamente fuori standard rispetto ad altri editor (o meglio, vista la sua importanza, `vi` costituisce uno standard a sé).

Doppia modalità operativa: comando e inserimento

Il programma è caratterizzato da due modalità operative differenti: la modalità "comando" e la modalità "inserimento". Nella prima è possibile impostare i comandi generali per la gestione dell'intero file (in ambiente `vi` il file caricato si chiama *buffer*), mentre nella seconda è possibile scrivere e modificare il testo.

Appena caricato `vi` ci troviamo in modalità comando, in cui possiamo muoverci all'interno del *buffer* utilizzando i tasti di spostamento del cursore (quelli con le "frecce") o i tasti h per spostare il cursore verso sinistra, j per spostarlo in basso, k per muoverlo verso l'alto e l per muoverlo a destra.

Dalla modalità comando il tasto i ci permette di passare alla modalità inserimento. Per tornare in modalità comando si deve battere il tasto Esc (che su alcuni sistemi è sostituito dalla sequenza Ctrl-[). Il programma non visualizza nessuna indicazione sulla modalità attiva, quindi si deve porre una certa attenzione quando si commuta da una modalità all'altra.

Entrati in modalità inserimento possiamo digitare il testo da inserire, proprio come su qualsiasi altro programma di videoscrittura. Ogni modifica, cancellazione o spostamento, avviene però sul singolo carattere: per operare su interi blocchi di testo ci si deve portare in modalità comando, dove si può, ad esempio, cancellare

una intera linea del *buffer* battendo due volte il tasto d, oppure il singolo carattere sotto al cursore, battendo x. Per tornare in modalità inserimento, invece del tasto i che ci permette di inserire il testo a partire dalla posizione attuale del cursore (*insert*), possiamo battere A (*add*) che ci permette di aggiungere il testo alla fine della linea su cui si trova il cursore.

Comandi per operare sul testo

La modalità comando mette a disposizione una vasta gamma di istruzioni, costituite da una o due lettere, con cui è possibile operare anche sull'intero *buffer*. Per utilizzarle si deve battere : (due punti) e poi digitare il comando. Vediamone alcuni:

`co` (*copy*) Per copiare dopo la riga n_3 il testo che va dalla riga n_1 alla riga n_2 si deve digitare il comando "`:`n_1, n_2`co`n_3";

`mo` (*move*) Per spostare dopo la riga n_3 il testo che va dalla riga n_1 alla riga n_2, si deve digitare il comando "`:`n_1, n_2`mo`n_3";

`d` (*delete*) Per cancellare il testo dalla riga n_1 alla riga n_2 si digiti il comando "`:`n_1, n_2`d`";

`r` (*read*) Per inserire il contenuto del file *nome* dopo la riga n, si digiti il comando "`:`n `r` *nome*";

`w` (*write*) Per salvare sul file *nome* il testo dalla riga n_1 alla riga n_2, si deve digitare il comando "`:`n_1, n_2 `w` *nome*"; per salvare l'intero file si digiti semplicemente "`:w`", oppure "`:w` *nome*";

`q` (*quit*) Per uscire dal programma si digiti "`:q`"; per uscire senza salvare su file le modifiche apportate al testo si digiti "`:q!`", mentre per uscire e salvare il testo su file si deve dare il comando "`:wq`" oppure si può battere due volte il tasto Z (maiuscolo).

Per spostarsi direttamente su una certa linea all'interno del testo si può digitare "`:`n", dove n è il numero di riga; per andare alla fine del file si può battere più semplicemente G.

Come in ogni editor che si rispetti, anche in `vi` è possibile cercare una certa stringa di caratteri all'interno del *buffer*: dalla modalità comando basta battere / (*slash*) seguito dalla stringa da cercare (o da una espressione regolare che descrive la stringa che stiamo cercando); una volta trovata la prima occorrenza della stringa, per cercarne un'altra occorrenza basterà battere nuovamente /, senza specificare nessuna stringa; si può anche battere il tasto n (*next*) per trovare la successiva occorrenza, oppure N per trovare l'occorrenza precedente.

Per utilizzare i comandi di `vi` è estremamente utile poter identificare i numeri di linea del testo inserito; per questo è opportuno, appena avviato il programma, dare il comando "`:set number`" che attiva la numerazione automatica delle linee. Per far sì che `vi` abiliti automaticamente questa funzione, si deve inserire il comando "`set number`" nel file "`.exrc`" che contiene i comandi che devono essere eseguiti da `vi` all'inizio della sessione di lavoro; come tutti i file di configurazione, anche questo comincia con un punto e deve essere inserito nella nostra *home directory*. A titolo di esempio riporto il contenuto del mio file "`.exrc`":

Numerazione delle righe, set number

Configurazione di vi, .exrc

```
set autoindent
set tabstop=2 shiftwidth=2
set number
set wm=10
```

Per aprire un file in modalità di sola lettura, senza correre il rischio di modificarlo inavvertitamente, si può utilizzare l'opzione "`-r`" o il comando `view`.

`vi` ha un gran numero di comandi, alcuni dei quali (pochissimi, a dire il vero) sono stati descritti in queste pagine; per avere un riferimento più dettagliato sulle varie possibilità offerte dal programma è opportuno riferirsi alla documentazione del proprio sistema.

3.2 Emacs

L'editor `emacs` è forse più sofisticato, ma altrettanto complesso e potente di `vi`. Supporta un linguaggio per la compilazione di *macro* molto simile al *Lisp*, un linguaggio funzionale che opera su liste. Questo linguaggio di macro è potente al punto che qualcuno si è anche divertito a sviluppare dei veri programmi (ed anche dei videogiochi) che girano all'interno dell'editor `emacs`!

Come struttura il programma si avvicina maggiormente agli editor tradizionali, in questo caso non c'è la doppia modalità inserimento/comandi, ma tutto può essere fatto in un unico ambiente e le sequenze di tasti che non devono essere inserite nel testo, perché servono a dare dei comandi al programma, sono tutte composte premendo Ctrl insieme ad altri tasti.

Una caratteristica importante di `emacs` è il fatto di poter operare contemporaneamente su più file (che anche in questo caso vengono chiamati *buffer*).

Esiste anche una versione grafica di `emacs`, denominata `xemacs`, che permette di lavorare in tutta comodità su un terminale grafico X11, utilizzando per i comandi, invece delle combinazioni di tasti, i consueti menù a tendina selezionabili con il mouse.

Spesso `emacs` è utilizzato proprio in questa modalità grafica guidata dai menù, che quindi non richiede ulteriori spiegazioni; ci limiteremo allora a descrivere i principali comandi che devono essere impartiti da tastiera nel caso si utilizzi il programma su un terminale alfanumerico. Generalmente si utilizzano i caratteri di spostamento del cursore per muoversi all'interno del buffer ed il tasto backspace per effettuare le cancellazioni, tuttavia non tutti i terminali prevedono l'uso di questi tasti; in tal caso è bene riferirsi al seguente elenco:

Combinazioni di tasti per operare sul testo

Ctrl-V avanti di una schermata;

Ctrl-v indietro di una schermata;

Ctrl-p indietro di una riga (*previous*);

Ctrl-n avanti di una riga (*next*);

Ctrl-b indietro (a sinistra) di un carattere (*back*);

Ctrl-f avanti (a destra) di un carattere (*forward*).

Per la gestione dei file si deve fare riferimento ai seguenti comandi:

Ctrl-x Ctrl-f *nomefile* Carica nel buffer il file specificato;

Ctrl-x Ctrl-s Salva su file il buffer corrente;

Ctrl-x Ctrl-c Termina l'esecuzione del programma; prima di uscire chiede se si desidera salvare su file eventuali buffer modificati, ma non ancora salvati.

Ogni volta che si usa il comando Ctrl-x Ctrl-f per aprire un nuovo file, lo si carica nel buffer, ma non si eliminano i buffer precedentemente caricati: semplicemente avremo più buffer aperti e distinti, ognuno contenente il testo di un file. Si può passare da un buffer all'altro utilizzando le seguenti combinazioni di tasti che gestiscono le "finestre" di `emacs`:

Ctrl-x Ctrl-b Elenca i *buffer* attivi;

Ctrl-x Ctrl-f *nomebuffer* Se il *buffer* è già stato caricato, lo visualizza nella finestra attiva, altrimenti prima carica il file con quel nome, lo inserisce in un nuovo *buffer* e lo visualizza nella finestra attiva;

Ctrl-x 2 Divide lo schermo a metà creando una seconda finestra;

Ctrl-x o Se lo schermo visualizza più di una finestra (attivata con Ctrl-x 2) passa da una finestra all'altra (*other*);

Ctrl-x 1 Visualizza sullo schermo solo la finestra attiva (quella in cui si trova il cursore).

Per effettuare delle ricerche sul testo di un *buffer* si deve battere Ctrl-s (*search*) e poi digitare la stringa di testo da cercare; per ripetere la ricerca della stessa stringa sarà sufficiente battere di nuovo Ctrl-s.

Help on-line e tutorial

Emacs è dotato di un potente sistema di *help on-line* e di un *tutorial*, una sorta di breve corso guidato sulle funzioni principali del programma. Per accedere all'help, ed avere istruzioni sull'uso di un certo comando, basta battere Ctrl-h c e poi la sequenza di caratteri su cui si vuole un aiuto. Ad esempio:

```
>> Ctrl-h c Ctrl-p

C-p runs the command previous-line
```

Per avere informazioni più complete su una certa sequenza di tasti si può usare il comando Ctrl-h k seguito dal comando di cui si vogliono le informazioni. `emacs` visualizza il testo di help in una finestra separata, quindi per ripristinare la finestra su cui stavamo lavorando, eliminando l'help, dovremo battere Ctrl-x 1.

Per eseguire il *tutorial* si deve battere Ctrl-h Ctrl-h t. Il *tutorial* viene visualizzato in una finestra separata e permette anche di sperimentare praticamente numerosi comandi.

3.3 Pico

Spenderemo soltanto poche parole per descrivere questo semplice editor che, come abbiamo già detto, viene anche utilizzato dal programma `pine` come editor per la compilazione dei messaggi di posta elettronica.

Ciò che distingue `pico` dagli altri due programmi visti in questo capitolo, è sicuramente il fatto che tutti i comandi che è possibile impostare vengono riportati sinteticamente in una specie di menù visualizzato sulle ultime due righe del terminale. Ogni comando viene impostato premendo Ctrl insieme ad un altro tasto. A differenza di `emacs`, `pico` dispone di un unico *buffer* e quindi può operare su un unico file alla volta. In ogni caso qualsiasi paragone tra questo programma ed uno degli altri due editor visti in precedenza è del tutto fuori luogo: si tratta di prodotti di classe completamente diversa, potenti e versatili `vi` ed `emacs`, estremamente limitato, ma assai intuitivo nell'uso, `pico`.

Riportiamo una breve spiegazione dei comandi principali:

Ctrl-c visualizza il numero di linea su cui si trova il cursore (*current position*);

Ctrl-g Visualizza le informazioni di *help* sul programma (*get help*);

Ctrl-j "Giustifica", riallineando i margini destro e sinistro delle righe, il paragrafo corrente (*justify*);

Ctrl-k Cancella la riga su cui si trova il cursore (*cut text*);

Ctrl-o Salva su file il testo contenuto nel *buffer* (*write out*);

Ctrl-r Inserisce il testo di un altro file nella posizione corrente del cursore (*read file*);

Ctrl-u Annulla l'ultimo comando eseguito (*undo*);

Ctrl-w Cerca una stringa di caratteri nel testo (*where is*);

Ctrl-x Esce dal programma (*exit*).

Per cancellare o spostare un intero blocco di testo è possibile utilizzare il comando di "selezione estesa": ci si posiziona all'inizio del testo da selezionare e si batte Ctrl-^; muovendosi poi con il cursore, si estende la selezione: il testo marcato viene visualizzato in *reverse*. A questo punto è possibile tagliare l'intera selezione con il comando Ctrl-k e poi eventualmente incollarlo (con Ctrl-u) nella nuova posizione del cursore.

Pico non offre altre funzionalità avanzate: quindi solo lo stretto indispensabile per scrivere un breve file di testo (come un messaggio di posta elettronica, ad esempio) e poco altro. Se dovessimo avere la necessità di eseguire operazioni più complesse sul nostro file, dovremo rivolgeci alla maggiore potenza (e complessità) di `emacs` e `vi`.
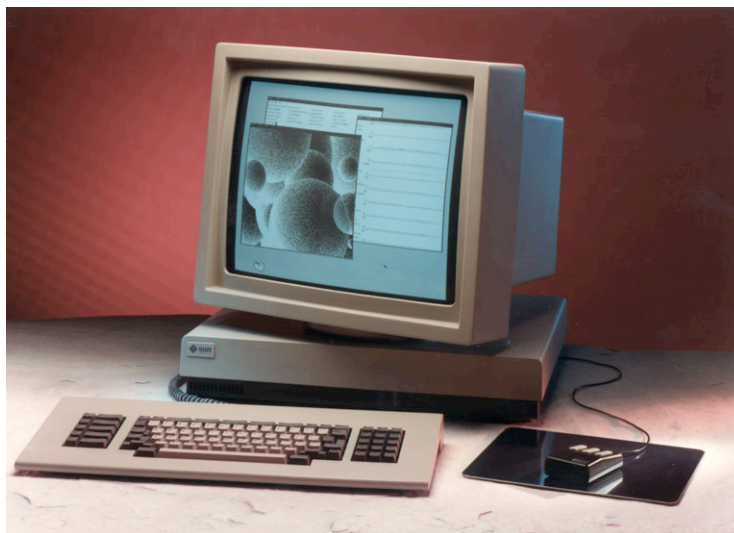
Figura 3.1: Workstation Sun 2-50

Capitolo 4

L'interfaccia grafica X Window

Unix nasce nei primi anni '70, quando i grandi elaboratori centralizzati, i *mainframe*, lavoravano soprattutto mediante schede perforate, ed i terminali con video e tastiera, così come li conosciamo oggi, erano appannaggio di pochi fortunati. L'intero sistema operativo è stato sviluppato quindi in modo tale da prescindere completamente dal tipo di terminale con cui l'utente avrebbe utilizzato il sistema. Dal momento che spesso l'utente di un sistema Unix non opera direttamente sulla *console* della macchina, ma è collegato con un terminale connesso al server attraverso una rete o una linea telefonica, si è cercato di limitare al massimo lo scambio di messaggi a video tra il sistema e l'utente, in modo da non penalizzare in modo eccessivo gli utenti dotati di una connessione lenta o con una capacità di banda ridotta. Negli ultimi dieci anni hanno avuto un grosso sviluppo le tecnologie per la gestione del video e della cosiddetta *interfaccia utente*, ossia quell'insieme di dispositivi hardware e software che permettono all'utente di comunicare (in gergo, di *interfacciarsi*) con la macchina. Grazie al lavoro svolto da un gruppo di ricercatori "visionari" (almeno per quei tempi, oggi possiamo dire lungimiranti) del Palo Alto Research Center (PARC) della Xerox, hanno avuto un particolare sviluppo quelle che oggi chiamiamo GUI, *Graphical User Interface*, che ci permettono di operare su un terminale grafico, interagendo col sistema principalmente mediante il mouse e l'uso estensivo di rappresentazioni grafiche (icone, finestre, la metafora del *desktop*, ecc.). Ci stiamo riferendo a quello che sicuramente ognuno di noi ha utilizzato spesso su un personal computer in ambiente Windows o su un Macintosh.

Anche Unix, ormai da diversi anni, è dotato di una propria interfaccia grafica standard, denominata *X Window* (o anche X11). In questo capitolo cercheremo di farci un'idea piuttosto sommaria ed un po' superficiale, di cosa ci offre in più l'uso di questo ambiente, rispetto al terminale alfanumerico.

4.1 X Window e i window manager

X Window è un sistema con una interessantissima architettura *client/server*, aperto, facilmente portabile da un sistema operativo ad un altro ed in grado di garantire anche un elevato livello di interoperabilità tra un sistema e l'altro. È necessario premettere tutto questo per poter spiegare la grande potenza di questo sofisticato sottosistema grafico, che adotta una filosofia completamente diversa da quella di altre interfacce utente grafiche strettamente legate ad uno specifico sistema operativo. L'interfaccia grafica di Microsoft Windows è strettamente legata al sistema operativo, come anche l'interfaccia di Mac OS X o del System della Apple: anzi,

il sistema operativo è composto anche dalle funzionalità della sua interfaccia grafica. X Window, invece, è un sottosistema grafico completamente indipendente dal sistema operativo: ne esiste dunque una implementazione (la più diffusa) per ogni sistema operativo Unix, ma ne esistono anche implementazioni per Windows e per i sistemi operativi della Apple, e in passato è stato utilizzato moltissimo anche in ambiente Digital VAX/VMS.

Architettura client/server di X Window

Il punto di forza di X11 è aver disaccoppiato la componente applicativa che richiede di utilizzare funzioni di visualizzazione ed interazione grafica con l'utente, dalla parte di presentazione che, sulla base delle richieste della componente applicativa, presenta sullo schermo del terminale grafico le icone e le finestre e consente di interagire con il sistema attraverso il mouse e la tastiera. È dunque un sistema *client/server* in cui la componente *client*, l'applicazione che produce un output grafico, si collega e colloquia con una controparte *server*, il cosiddetto X Window Server, che autorizza il collegamento e gestisce la visualizzazione dell'output sfruttando al meglio le caratteristiche del terminale grafico che controlla. Anche nei sistemi operativi Windows esiste una comunicazione tra l'applicazione e il sistema operativo che offre servizi di visualizzazione grafica, ma tale relazione viene stabilita attraverso delle funzioni API (*Application Programming Interface*) che permettono di visualizzare l'output solo sullo schermo del personal computer su cui sia l'applicazione che il sistema operativo vengono eseguiti. In ambiente X Window invece, le due componenti possono operare anche su macchine fisicamente distinte e distanti fra loro. Non a caso la maggior parte dei server Unix non sono dotati neanche di una componente hardware per la gestione della visualizzazione grafica (la cosiddetta "scheda grafica"), mentre questa è una componente indispensabile di ogni PC: il server Unix infatti si limita soltanto ad eseguire le applicazioni lanciate dagli utenti, mentre della visualizzazione dell'output in forma grafica si occupa un terminale grafico dotato di un server X Window. In questo modo diventa possibile utilizzare in modalità grafica la potenza di calcolo e di storage di uno stesso server da parte di più utenti connessi a terminali grafici distinti; ma al tempo stesso è anche possibile visualizzare sullo stesso terminale grafico l'output di applicazioni che girano su server differenti.
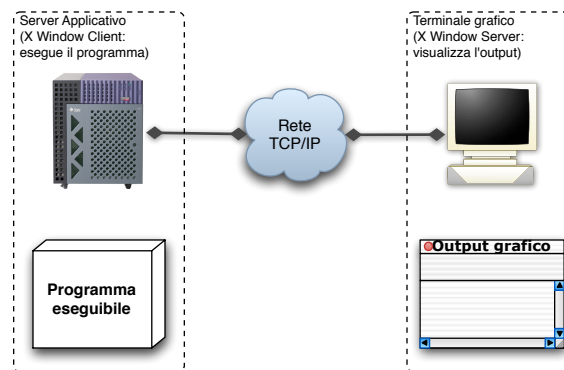
Figura 4.1: Schematizzazione dell'architettura client/server di X Window

Il sistema di gestione delle finestre, window manager

Un'altra componente importante del sottosistema grafico X Window è costituito dal *window manager* che, interagendo con il server X Window, si occupa di gestire l'aspetto grafico delle finestre e la modalità di interazione dell'utente con il *desktop*. Di fatto i "bottoni" con cui possiamo chiudere una finestra o ridurla ad una icona dislocata da qualche parte sul *desktop*, la "barra del titolo" con cui possiamo trascinare la finestra in giro per lo schermo ed altri aspetti grafico/funzionali che siamo

abituati a trovare su ogni interfaccia utente grafica, non fanno parte dell'applicazione, ma sono servizi offerti all'applicazione dal *window manager*. Su Microsoft Windows il *window manager* è uno solo ed è integrato nell'interfaccia grafica, mentre su X Window è possibile definire quale *window manager* si intende utilizzare, senza che questo comporti la necessità di apportare delle modifiche alle applicazioni. Uno stesso sistema può quindi essere configurato in modo tale da consentire ai propri utenti di scegliere il loro *window manager* preferito, con cui lavorare durante le sessioni al terminale grafico.

Alcuni *window manager* consentono la gestione di *virtual desktop*: lo schermo a disposizione dell'utente viene reso virtualmente molto più grande di quello fisicamente visibile attraverso il monitor; mediante una finestra particolare (chiamata *pager*) che rappresenta in scala ridotta l'intero grande schermo, è possibile selezionare la zona da "inquadrare" nel video del terminale.

I *window manager* storicamente più diffusi sono quattro, ma molti altri ne esistono o sono in via di sviluppo o di definizione:

- *Tab Window Manager* (TWM), il primo e forse il più "spartano" fra i *window manager*; non è molto sofisticato e tuttavia spesso è apprezzato proprio per la sobrietà con cui visualizza le finestre. Di default TWM si limita a visualizzare una barra del titolo sopra ad ogni finestra con un bottone per "iconizzare" la finestra stessa ed un altro per modificarne la dimensione; per spostare la finestra la si deve trascinare con il mouse *clickando* sulla barra del titolo. Esiste una versione di TWM che consente la gestione del *desktop* virtuale, denominata TVWM.

- *F(?) Virtual Window Manager* (FVWM), è uno dei più diffusi *window manager* per X Window in ambiente Linux e nasce come evoluzione di TWM. Oltre alla barra del titolo, FVWM visualizza anche un bordo intorno alle finestre e dà al tutto un aspetto tridimensionale. È possibile inserire diversi bottoni sulla barra del titolo a cui associare funzionalità diverse: rendere massima la dimensione della finestra, ridurla ad icona, ecc. Il bottone nell'angolo in alto a sinistra permette di accedere ad un menù per la gestione della finestra stessa. Con FVWM è disponibile la gestione dello schermo "virtuale" più grande di quello reale.

- *Open Look Window Manager* (OLWM), è il *window manager* che Sun Microsystems ha sviluppato inizialmente solo per le proprie *workstation*, ma che successivamente è stato portato, almeno in parte, su ogni altro sistema. Tra i *window manager* "storici" è uno dei più eleganti e sicuramente ha un livello di sofisticazione superiore a quello offerto dagli altri prodotti, cosentendo anche l'implementazione di funzionalità di "*drag-and-drop*" non presenti in twm ed fvwm. Degni di nota sono i menù a tendina "staccabili" (successivamente implementati anche da molti altri *window manager*) che è possibile collocare sul *desktop* nella posizione più comoda per l'utente. Esiste anche una versione di Open Look dotata di *desktop* virtuale (OLVWM).

- *Motif Window Manager* (mwm), è in assoluto il *window manager* più diffuso sulle implementazioni "proprietarie" (non *open source*) dei sistemi operativi Unix; ne è stata prodotta una versione compatibile anche per il mondo *open source*. Graficamente ricorda molto fvwm.

A questi *window manager* "storici"[1] si sono affiancati negli anni altri prodotti meno diffusi, ma più sofisticati, a cui si sono aggiunti dei prodotti più ampi, denominati *desktop manager*, che oltre ad offrire dei servizi per la gestione delle finestre

[1] Una rassegna piuttosto completa, con numerose immagini delle schermate dei diversi *window manager*, si trova sulla rete Internet sul sito web `http://xwinman.org`.

alle applicazioni grafiche, offrono all'utente una serie di *utility* che rendono più confortevole la gestione del desktop. Il più famoso in questa categoria è l'ambiente CDE (*common desktop environment*), supportato da tutti i principali Unix proprietari (Solaris, AIX, HP-UX, ecc.). Gnome e KDE (*K Desktop Environment*) sono degli ambienti ancor più sofisticati nati nel mondo *open source* che consentono agli sviluppatori di applicazioni di sfruttare delle librerie di funzioni grafiche aggiuntive, per realizzare programmi con interfacce utente ancora più complesse.
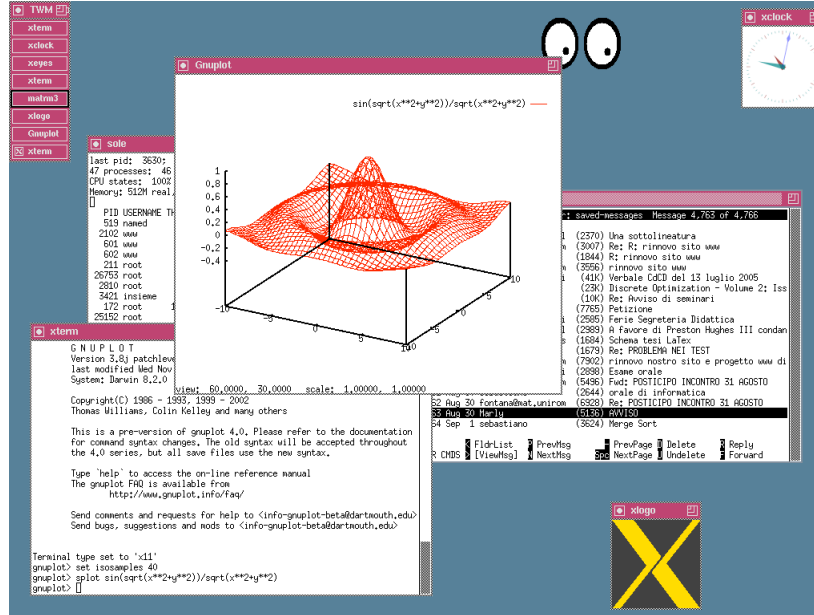
Figura 4.2: Una sessione X Window con TWM *window manager*

4.2 Utilizzo del display grafico "remoto"

Per sfruttare le capacità client/server del sistema X Window è necessario comunicare all'applicazione client le "coordinate" del display grafico e al server X11 l'indirizzo di rete del client autorizzato a sfruttare il display da remoto. Tutto questo viene impostato automaticamente con la configurazione di *default* se si utilizza la stessa *console* grafica della macchina Unix su cui stiamo lavorando (ad esempio quando si lavora "in locale" su un personal computer in ambiente Linux), ma deve invece essere definito esplicitamente dall'utente nel caso in cui intenda lanciare dei client grafici da una macchina remota.

Identificazione del display grafico per le applicazioni client, DISPLAY

La variabile di ambiente `DISPLAY` viene utilizzata dalle applicazioni client X11 per conoscere l'indirizzo di rete del server X Window e i riferimenti del display su cui deve essere inviato l'output grafico. Questa informazione viene specificata indicando l'indirizzo di rete IP o l'*hostname* del server X11 (della *workstation* grafica o dell'X-Terminal su cui opera l'utente), il numero del terminale grafico collegato alla *workstation* ed il numero dello schermo tra quelli collegati ad uno stesso terminale. Tipicamente una *workstation* Unix ha un terminale grafico (*display*) a cui è collegato un solo schermo (*screen*), ma in configurazioni più complesse (ad esempio per utenti che si occupano di computer grafica o nelle sale di controllo in cui sono presenti moltissimi schermi collegati ad una stessa postazione di lavoro) è possibile collegare più dispositivi hardware per consentire a molti

utenti di operare sullo stesso terminale grafico. Il valore della variabile `DISPLAY` deve essere definito con la seguente sintassi: "*hostname*:*display*.*screen*", ad esempio con il comando "`export DISPLAY=woodstock:0.0`" nella *shell* Bash, o con "`setenv DISPLAY woodstock:0.0`" utilizzando la C Shell. Il valore per i parametri *display* e *screen* è costituito da un numero intero progressivo, a partire da 0 che identifica le unità di *default*.

Al tempo stesso è necessario comunicare al server grafico X Window che può accettare richieste di visualizzazione da parte delle applicazioni client che operano su un determinato sistema; il server X11 è dotato infatti di un criterio di protezione che impedisce a chiunque di visualizzare qualcosa sullo schermo di un server grafico, altrimenti diventerebbe ben presto impossibile lavorare su quello schermo. Il comando per configurare i criteri di protezione dell'X server durante la sessione di lavoro in corso è `xhost`. Con questo comando si può comunicare al server X11 da quali indirizzi di rete IP può accettare connessioni e da quali invece deve rifiutarli. Di *default* la configurazione prevede di rifiutare qualsiasi connessione tranne quelle provenienti dal sistema locale (*localhost*); con i comandi "`xhost +`*hostname*" e "`xhost -`*hostname*" si configura il sistema per accettare e per rifiutare le connessioni provenienti dai client che operano sulla macchina identificata dal nome *hostname*.

Permessi di accesso al server X11, xhost

Vediamo un esempio piuttosto elementare, ma che dovrebbe aiutare a chiarire eventuali dubbi. Supponiamo di operare su una *workstation* Unix identificata dall'*hostname* "woodstock", dotata di *console* grafica e dunque anche di un server X Window. Supponiamo anche di voler eseguire su un'altra macchina Unix "remota" (ossia dislocata in una posizione distinta da quella della *workstation* di fronte a cui ci troviamo), identificata dall'hostname "aquilante", una certa applicazione grafica (ad esempio il programma `xlogo`) utilizzando però come terminale (tastiera, mouse, schermo e capacità grafiche ad alta risoluzione) la *workstation* "woodstock". Allora, aprendo una *shell* su "woodstock" (server X11), digiteremo il seguente comando per consentire a "aquilante" di connettersi in modalità grafica:

```
marco@woodstock ~$ xhost +aquilante
marco@woodstock ~$ _
```

Quindi, dopo aver aperto una *shell remota* su "aquilante" (vedremo nel prossimo capitolo come si può effettuare questa operazione), imposteremo le "coordinate" del display grafico ed eseguiremo l'applicazione con i seguenti comandi:

```
liverani@aquilante ~$ export DISPLAY=woodstock:0.0
liverani@aquilante ~$ xlogo &
liverani@aquilante ~$ _
```

Il programma `xlogo` viene eseguito sulla macchina remota "aquilante", impegnando la memoria, la CPU, le risorse di calcolo di quella macchina, ma il suo output e l'interazione con l'utente attraverso le finestre, il mouse e la tastiera, viene effettuato sulla macchina locale "woodstock".

4.3 Xterm

Mediante l'uso di X Window si può sperimentare e trarre profitto con maggiore facilità dalla grande potenza del *multitasking* di Unix. Spesso utilizzeremo il terminale grafico per compiti che avremmo potuto svolgere ugualmente su un normale terminale alfanumerico, ma che grazie ad X Window possiamo svolgere con maggiore comodità ed efficacia. Uno dei programmi più usati sotto X11 è `xterm`, una finestra

di emulazione di terminale, dove viene eseguita la *shell* da cui è possibile lanciare altri programmi. Sostanzialmente `xterm` riproduce le funzionalità di un terminale alfanumerico, in una finestra che però può essere molto più ampia delle 24 righe da 80 caratteri ciascuna tipiche di un terminale VT100. Alle funzioni di base di un terminale alfanumerico, `xterm` aggiunge numerose funzionalità rese disponibili dall'ambiente grafico X Window, come la possibilità di scorrere a ritroso il testo della sessione di lavoro mediante una *scroll bar*, la possibilità di stabilire arbitrariamente il tipo di carattere (il *font*) con cui visualizzare il testo nella finestra, e le funzioni di "copia & incolla".

Copia & incolla di testo sotto X Window

In particolare la funzione di "copia & incolla" (*copy & paste*) tipica di tutte le interfacce utente grafiche, è una caratteristica gestita dal server X11 e non dalle singole applicazioni. In pratica consente di selezionare con il mouse (evidenziandola in *reverse*) una porzione di testo visualizzata in una finestra qualsiasi (non solo in una finestra di `xterm`), copiandola in un'area di memoria della macchina denominata *clipboard*; in un secondo momento è possibile incollare il testo presente nel *clipboard* in un'altra finestra o in un'altra posizione della stessa finestra da cui era stata copiata. Per copiare una porzione di testo è sufficiente selezionarla con il mouse, utilizzando il pulsante sinistro; una volta effettuata una selezione, la si può estendere selezionando altre parti del testo contigue con il tasto destro del mouse. Per incollare (*paste*) altrove il testo selezionato basta premere il pulsante centrale.[2]

Aprendo più finestre di `xterm` sullo schermo, potremo operare contemporaneamente, grazie al *multitasking*, su più applicazioni. Dalla *shell* attivata nella finestra di `xterm` è possibile lanciare anche altre applicazioni grafiche indipendenti dalla finestra stessa.

In genere quando si apre una sessione di lavoro su un terminale grafico X Window (spesso chiamato più semplicemente X) viene attivata automaticamente una finestra `xterm`. In alcuni casi la prima finestra di `xterm` aperta esegue la cosiddetta "*shell* di login": quindi chiudendo questa sessione di `xterm` (e dunque terminando la *shell* di login) si effettua il *logout* dal sistema.

La *shell* che ci viene resa disponibile mediante `xterm` sarà il nostro programma chiave per controllare il sistema durante la sessione di lavoro al terminale grafico. Cerchiamo di entrare un po' più in dettaglio cominciando con qualche esempio. Per sperimentare subito il *multitasking* potremmo provare ad aprire un altro `xterm`, digitando semplicemente il comando "`xterm`" al *prompt* della *shell*. Dopo pochi istanti il sistema apre una seconda finestra con un'altra *shell* attiva al suo interno. Ci accorgiamo subito però che la *shell* della prima finestra non è più attiva, ovvero sembra "congelata" in uno stato in cui non viene più presentato il *prompt* e dunque non si riesce ad immettere altri comandi. In effetti la *shell* è stata proprio "congelata" temporaneamente per poter eseguire l'applicazione che da essa è stata lanciata (il secondo `xterm`). Dove è finito il *multitasking*? La risposta è semplice: non l'abbiamo sfruttato a dovere. Portando il mouse all'interno della finestra `xterm` appena attivata, digitiamo il comando "`exit`" e vediamo ricomparire il *prompt* della *shell* che fino ad ora era rimasta inattiva. Come avevamo visto nel paragrafo 2.6 a pagina 23 è necessario aggiungere il simbolo "`&`" alla fine della linea di comando per attivare il processo "figlio" in una sessione separata, sganciata da quella del processo "padre". Quindi per lanciare un secondo `xterm` si deve digitare il comando "`xterm &`": in questo modo viene aperta una seconda finestra e viene attivata una *shell* al suo interno senza "congelare" l'altra *shell*. Ogni volta che da un `xterm` vorremo lanciare un'applicazione che deve essere eseguita in una finestra separata, dovremo ricordarci di aggiungere il simbolo "`&`" alla fine della linea di comando. I comandi che digiteremo sulla tastiera verranno indirizzati alla finestra attiva (quella

Lanciare applicazioni in altre finestre, &

[2] I terminali grafici X Window hanno un mouse dotato di tre pulsanti; sulle *workstation* su cui il mouse è dotato soltanto di due tasti, premendoli insieme contemporaneamente si simula la pressione del tasto mancante, il tasto "centrale" del mouse.

in cui si trova il mouse), evidenziata in modo opportuno dal *window manager*; a differenza di quanto avviene su Microsoft Window, la configurazione di *default* dei *window manager* tradizionali non prevede che si debba selezionare con il pulsante sinistro del mouse una finestra per renderla attiva: è sufficiente che il puntatore del mouse si trovi nell'area della finestra per renderla attiva; in questo modo è possibile digitare dei comandi in una certa finstra di `xterm`, anche quando questa è parzialmente coperta da altre finestre che preferiamo tenere in primo piano.

Selezione della finestra attiva

Non sempre però è opportuno eseguire un'applicazione in una sessione separata: ad esempio se volessimo usare l'editor `vi` all'interno di una finestra `xterm`, non dovremo aggiungere la "`&`" alla fine della riga di comando, proprio perché questa volta desideriamo interrompere momentaneamente l'esecuzione della *shell*, che deve lasciare il posto, all'interno della stessa finestra, all'editor. Sperimentare direttamente queste differenze aiuterà a capire questi meccanismi molto più di quanto non possano fare queste intricate spiegazioni.

È possibile lanciare il programma `xterm` specificando una serie di opzioni, alcune delle quali sono tipiche di tutte le più diffuse applicazioni grafiche che operano sotto X. Vediamone alcune, che in certi casi potremo utilizzare anche con altri programmi:

Opzioni sulla linea di comando di xterm

`-geometry` serve per specificare la dimensione e la posizione iniziale della finestra che si sta aprendo; ad esempio il comando "`xterm -geometry 90x30+100-20 &`" apre un `xterm` di 90 colonne per 30 righe, distante 100 pixel dal margine sinistro dello schermo e 20 dal margine inferiore;

`-bg` specifica il colore di *background* della finestra che si sta aprendo; ad esempio: "`xterm -bg skyblue4 &`"; l'elenco dei colori di default, generalmente è contenuto nel file "`/usr/X11/lib/X11/rgb.txt`";

`-fg` specifica il colore di default per il testo visualizzato nella finestra (*foreground*), ad esempio: "`xterm -bg gray20 -fg white &`";

`-cr` specifica il colore del cursore, ad esempio: "`xterm -bg skyblue4 -fg white -cr yellow &`";

`-fn` specifica il nome del *font* (tipo di carattere) da utilizzare, ad esempio: "`xterm -fn lucidasanstypewriter-12 &`";

`-sb` abilita la visualizzazione della *scroll bar*, ad esempio: "`xterm -sb &`".

Si possono modificare alcune opzioni del programma `xterm` anche quando questo è già in esecuzione (*run time*). Tenendo premuto il tasto Ctrl è possibile attivare tre diversi menù selezionando i tre pulsanti del mouse sulla finestra:

Menù di opzioni di xterm

- *Main Options* – si abilita questo menù premendo il pulsante sinistro del mouse; serve fondamentalmente per dirigere l'input da tastiera esclusivamente a quella finestra, anche quando non è la finestra attiva, per ridisegnare il contenuto della finestra, per uscire dal programma (ma è meglio usare il comando `exit` impostato da tastiera);

- *VT Options* – si attiva mediante il pulsante centrale del mouse; serve principalmente per abilitare/disabilitare la scroll bar, o per visualizzare la finestra in reverse (con i colori del background e del foreground invertiti);

- *VT Fonts* – si attiva con il pulsante destro del mouse e serve per stabilire la dimensione del font utilizzato.

4.4 Alcune utility

Lavorare in un ambiente grafico come X Window ci permette di utilizzare tante comode *utility*, che, pur non essendo indispensabili per svolgere il nostro lavoro, lo semplificano rendendo l'ambiente della sessione più "confortevole". In questa sezione descriviamo brevemente alcuni di questi comandi; come al solito, per ottenere informazioni più dettagliate è bene riferirsi alle pagine di manuale, magari mediante il programma `xman`.

Colore dello sfondo del desktop, xsetroot

Con `xsetroot` si possono impostare alcuni parametri di configurazione per lo sfondo dello schermo (la cosiddetta *root window*). In particolare l'opzione "`-solid`" permette di impostare il colore dello sfondo; ad esempio il comando "`xsetroot -solid steelblue`" imposta un colore azzurro chiaro.[3] Sui terminali monocromatici potranno invece essere utili le opzioni "`-gray`" e "`-def`" che impostano due diversi sfondi in bianco e nero. Con l'opzione "`-bitmap`" è possibile specificare il nome di un file che contiene una immagine in formato *bitmap* in bianco e nero, che viene visualizzata sullo sfondo ripetendola fino a ricoprire l'intero schermo. Le immagini *bitmap* possono essere disegnate utilizzando il programma `bitmap`.

Visualizzazione dell'orologio, xclock, oclock, clock

Esistono numerosi orologi visualizzabili in una finestra sotto X: il più diffuso è `xclock`, ma spesso si trovano anche `oclock` e `clock`, quest'ultimo sotto Open Look. Sono numerose le opzioni che possiamo specificare per modificare l'aspetto degli orologi. Con `xclock` ad esempio l'opzione "`-update 1`" visualizza anche la lancetta dei secondi, mentre "`-chime`" abilita il segnale acustico emesso ogni ora; con `oclock` l'opzione "`-transparent`" rende trasparente lo sfondo dell'orologio, provocando un divertente effetto visivo.

Calcolatrice, xcalc

La calcolatrice, sempre presente sotto X, può essere richiamata con il comando `xcalc`; se viene richiamata con l'opzione "`-rpn`" viene abilitata la "notazione polacca inversa" (*reverse polish note*), tipica di alcune calcolatrici tascabili HP.

Alcuni programmini divertenti, ma completamente inutili, sono `xeyes`, che visualizza due occhi che seguono con lo sguardo i movimenti del puntatore del mouse, `xlogo`, che visualizza il logo di X Window, e `xmelt` che provoca uno spettacolare quanto innocuo effetto di "squagliamento" dello schermo del terminale: se è presente sul vostro sistema potrà esservi utile... per stupire gli amici!

Screen saver, blocco del terminale, xset, xlock

X Window è dotato di una funzionalità di *screen saver* che annerisce completamente lo schermo del terminale dopo diversi minuti di inattività dell'utente. Può essere configurato con il comando `xset` che accetta, tra l'altro, diversi parametri proprio per la gestione dello *screen saver*. Ad esempio l'opzione "`s activate`" consente di attivare immediatamente lo screen saver. Con l'opzione "`s blank`" si indica che lo *screen saver* deve produrre uno schermo completamente nero, mentre "`s noblank`" produce un "salva-schermo" che elimina ogni finestra dallo schermo e visualizza il logo di X Window. L'opzione "`s` *n m*" consente di specificare che lo *screen saver* deve attivarsi dopo *n* secondi di inattività dell'utente e che ogni *m* secondi deve essere cambiata la posizione in cui viene visualizzato il logo di X11; nel seguente esempio viene impostata l'attivazione dello *screen saver* dopo cinque minuti (300 secondi) di inattività dell'utente:

```
$ xset -s noblank -s 300 10
$ _
```

Spesso può essere utile attivare immediatamente uno *screen saver* bloccando il terminale con la nostra *password*, quando ci allontaniamo per qualche minuto dal nostro posto di lavoro. Il comando `xlock` abilita questa funzionalità visualizzando sullo schermo coloratissimi "effetti speciali"; se viene specificata l'opzione

[3] Come abbiamo già accennato nelle pagine precedenti, l'elenco dei nomi dei colori che possono essere utilizzati si trova nel file di testo `/usr/X11/lib/X11/rgb.txt`.

"`-nolock`" non sarà necessario digitare la *password* dell'utente per sbloccare il terminale (basterà un semplice *click* del mouse). Con l'opzione "`-mode`" è possibile scegliere l'animazione da visualizzare che altrimenti viene selezionata a caso tra quelle disponibili (hop, life, qix, image, swarm, rotor, pyro, flame, worm, random ed altre ancora). Ad esempio si può attivare lo screen saver digitando "`xlock -nolock -mode flame`".

Oltre a queste divertenti utility, di solito ne vengono fornite altre un po' più sofisticate ed utili, come ad esempio gli editor `xedit` e `textedit`. Il primo è assai semplice e sfrutta la funzionalità di "copy & paste" offerta da X Window. È possibile scorrere il testo del file caricato mediante la *scoll bar* ed è possibile posizionare il cursore nella finestra anche con il mouse; tuttavia è un editor molto primitivo a cui mancano numerose funzionalità fondamentali. `textedit` è invece l'editor di sistema dell'ambiente Open Look. È un programma piuttosto comodo e abbastanza sofisticato, meno potente di `emacs` e `vi`, ma pur sempre un utile strumento di lavoro. Dispone delle classiche funzionalità di *cut, copy & paste*, e di ricerca e sostituzione di stringhe di caratteri all'interno del testo; tutte queste funzionalità sono accessibili mediante i menù a tendina ed alcune *dialog box*.

Altri editor a finestre, xedit, textedit

4.5 Altre applicazioni grafiche

In questa sezione diamo un rapido sguardo ad alcune applicazioni molto diffuse ed assai utili, che necessitano dell'ambiente grafico X Window per poter essere eseguite. Iniziamo con `ghostview`, già citato nel paragrafo 2.4, che serve per visualizzare e stampare file in formato PostScript (il cui nome, in genere, termina con "`.ps`"). Il programma è guidato mediante dei menù selezionabili con il mouse e permette sostanzialmente di scorrere le pagine del testo visualizzandole in una finestra grafica, selezionarle tutte o solo una parte, stampare l'intero documento o solo le pagine selezionate. Il formato PostScript riveste una particolare importanza in ambiente Unix perché è un sofisticato formato standard utilizzabile su piattaforme hardware/software completamente diverse, ma sufficientemente potente tanto da includere diversi stili del testo e figure in formato grafico. Tramite la rete Internet si possono reperire facilmente numerosi documenti e manuali tecnici realizzati in PostScript.

Visualizzazione e stampa di documenti in formato Postscript, ghostview

Oltre a questo formato, ne esiste anche un altro, denominato *DVI* (Device Independent), che garantisce la "portabilità" di documenti dotati di una formattazione ricca e complessa su piattaforme diverse. È questo il formato in cui vengono prodotti i documenti scritti e compilati con il programma TEX, un sofisticato strumento di composizione tipografica molto usato in ambito scientifico ed universitario. Per visualizzare i file DVI è spesso utile il programma `xdvi`, che però, al contrario di `ghostview`, non permette di stampare il file, ma soltanto di mostrarlo a video e di scorrerne le pagine. Se non è stato predisposto un apposito filtro di stampa per convertire automaticamente i file DVI in un formato adatto alla nostra stampante, con il programma `dvips` potremo convertire il file in formato DVI in un file equivalente in formato PostScript. Il programma `xtex`, utile soprattutto a chi utilizza il TEX per scrivere i propri documenti, al pari di `xdvi` consente di visualizzare i file DVI, ma in più gestisce anche la stampa.

Documenti in formato DVI, xdvi

Negli ultimi anni si è affermato come formato portabile universale per documenti elettronici il formato PDF (*Portable Document format*) di Adobe. Per visualizzare e stampare documenti in formato PDF si possono utilizzare il programma `xpdf` o il più sofisticato Acrobat Reader della stessa Adobe, che viene lanciato dal comando `acroread`.

Documenti PDF, xpdf, acroread

Oltre al PostScript, al DVI e al PDF, orientati per lo più al testo, esistono numerosissimi formati standard per la memorizzazione di immagini grafiche di qualità

Visualizzazione e fotoritocco di immagini grafiche, xv, display

fotografica. I più diffusi sono il *GIF*, il *BMP* ed il *JPEG*. Per visualizzare in una finestra del nostro terminale grafico un'immagine codificata in uno di questi formati, su molti sistemi è disponibile il programma `xv`. Mediante le numerose funzioni offerte dal pannello di controllo attivabile *clickando* col pulsante destro del mouse sulla finestra principale del programma, è possibile anche modificare e rielaborare l'immagine caricata, per poi salvarla, magari in un altro formato. Una interessante funzione offerta da `xv` è quella che permette di "catturare" il contenuto di un'altra finestra visualizzata sul *desktop* per poi salvarla o rielaborarla. Per la visualizzazione e la rielaborazione di immagini grafiche ad alta risoluzione sono molto utili i tool della suite *open source* ImageMagick, come ad esempio il comando `display` che consente di visualizzare e ritoccare immagini.

Grafici di funzioni matematiche, gnuplot

L'ultima applicazione descritta in questa brevissima panoramica è `gnuplot`, un famoso programma per la visualizzazione di grafici di funzioni in una o due variabili (grafici di curve o di superfici) e per la rappresentazione grafica (mediante interpolazione lineare) di dati provenienti da elaborazioni esterne (ad esempio una collezione di dati raccolti per via sperimentale). `gnuplot` opera su due finestre distinte: quella in cui è possibile impartire i comandi, che è la finestra `xterm` da cui si è lanciata l'applicazione (senza "`&`", quindi) e quella in cui avviene la visualizzazione dei grafici. Sono numerosi i comandi che è possibile impartire da tastiera all'interno di `gnuplot`, riporto brevemente solo i principali, rimandando come al solito per una descrizione più approfondita alla documentazione e all'*help* in linea che è possibile richiamare con il comando `help`.

`plot` [x_1:x_2] $f_1(x), f_2(x), \ldots$ Per visualizzare il grafico delle funzioni $f_1(x), f_2(x), \ldots$ nell'intervallo $[x_1, x_2]$;

`plot` *file* Per visualizzare il grafico per punti, prendendo i valori delle ordinate da un file di dati; con il comando "`plot` *file* `with lines`" si rappresenta lo stesso grafico, ma i punti vengono anche collegati mediante dei segmenti;

`splot` [x_1:x_2] [y_1:y_2] $f(x, y)$ Per visualizzare il grafico della superficie rappresentata dalla funzione di due variabili $f(x, y)$ nel dominio $[x_1, x_2] \times [y_1, y_2]$;

`hidden3d` Per visualizzare il grafico delle superfici in modo non trasparente, ma "solido", nascondendo le linee nascoste;

`nohidden3d` Per visualizzare il grafico delle superfici in modo trasparente, mostrando anche le linee nascoste.

`quit` Per uscire da `gnuplot` e tornare alla *shell*.

4.6 Configurazione della sessione grafica

File di configurazione della sessione X Window

Ogni singolo utente del sistema può configurare e personalizzare la propria sessione grafica in ambiente X Window operando su alcuni file di configurazione presenti nella propria *home directory*. Come al solito si tratta di semplici file di testo o *shell script*, che, rispettando una sintassi molto elementare, consentono a chiunque di modificare l'aspetto e il comportamento dell'ambiente grafico. Dal momento che ogni applicazione in ambiente X11 ammette numerosissime opzioni per personalizzarne l'output, è molto comodo poterle definire una volta per tutte in un file di configurazione, senza doverle specificare ogni volta sulla linea di comando.

La configurazione dell'ambiente da parte dell'utente avviene principalmente attraverso i seguenti file:

`.xsession` per la configurazione iniziale della sessione in ambiente grafico, per quei sistemi in cui il login avviene attraverso il sistema grafico XDM (*X Display Manager*);

`.xinitrc` per la configurazione iniziale della sessione in ambiente grafico, per quei sistemi in cui la sessione grafica in ambiente X Window viene lanciata manualmente dall'utente (il comando varia da sistema a sistema: `startx`, `xstart`, `openwin`, ecc.);

`.Xresources` per la configurazione delle opzioni di default per le principali applicazioni grafiche;

`.mwmrc`, `.twmrc`, `.fwmrc` per la configurazione del *window manager* selezionato per la gestione delle finestre.

Configurazione iniziale della sessione di lavoro, .xsession, .xinitrc

I file "`.xsession`" e "`.xinitrc`" sono degli *shell script* che contengono i comandi da eseguire ogni volta immediatamente dopo l'avvio della sessione grafica X Window. Con questi *script* di solito viene lanciato un `xterm` (per consentire all'utente di iniziare a lavorare), viene impostato l'ambiente grafico (colore dello sfondo, configurazione dello *screen saver*) e viene lanciato un *window manager*. La sintassi dei due file è la stessa; di seguito riportiamo un esempio molto semplice:

```
#!/bin/sh
#   ~/.xsession: file di configurazione della sessione X
#
xrdb -merge .Xresources
xset s 300 15 s noblank
xsetroot -solid LightSkyBlue4
xterm &
xclock -geometry -10+10 &
exec twm
```

Configurazione delle applicazioni grafiche, xrdb, .Xresources

Il file "`.Xresources`" viene utilizzato per definire le impostazioni di default di alcune delle applicazioni grafiche; per caricare le impostazioni presenti nel file si utilizza il comando `xrdb` che, con l'opzione "`-merge`" consente di aggiungere (ed eventualmente sovrapporre) le nuove impostazioni a quelle di default. Il file contiene una opzione per ogni riga: viene specificato il nome del programma a cui si applica l'opzione seguito da un asterisco e dall'impostazione che si intende definire. Di seguito riportiamo un esempio che può servire a rendere l'idea; l'elenco delle opzioni (chiamate anche "risorse") è disponibile nella pagina di manuale di ogni programma:

```
xman*topBox: False
xman*bothShown: true
XTerm*scrollBar: True
XTerm*saveLines: 300
xclock*width: 100
xclock*height: 100
xclock*update: 1
xclock*clientDecoration: all
xeyes*height: 66
xeyes*width: 100
```

Configurazione del window manager

Per quanto riguarda la configurazione dei singoli *window manager* si rimanda infine alla documentazione (e alle pagine di manuale) degli stessi programmi. Mediante i file di configurazione sopra indicati è possibile definire il colore del bordo delle finestre attive e non attive, il colore e l'aspetto dei "menù a tendina", il comportamento dei pulsanti del mouse quando vengono utilizzati sullo sfondo del desktop o sulla barra del titolo delle finestre, e molto altro ancora.

Figura 4.3: Una *workstation* grafica NeXT Station

Capitolo 5

Alcuni strumenti per l'uso della rete Internet

I sistemi Unix sono spesso inseriti in un ambiente di rete TCP/IP, dove vengono utilizzati da numerosi utenti connessi da terminali locali o remoti; in molti casi sono proprio dei server in ambiente Unix a gestire servizi importanti come la posta elettronica e la gestione di siti web sulla rete Internet. In effetti un sistema Unix si integra alla perfezione in un ambiente di connettività vasta ed eterogenea come quello di Internet, dove devono comunicare tra loro, in modo del tutto trasparente agli utenti, macchine e sistemi operativi anche molto diversi.

5.1 La Rete delle reti

Origini e crescita vertiginosa della rete Internet

Originariamente Internet nasce con il nome di *ARPANET*, un progetto del Ministero della Difesa americano, che doveva definire un modello di rete di telecomunicazioni inattaccabile dal nemico. Siamo nei primi anni '60, in piena guerra fredda, l'esercito americano è ossessionato dal pensiero di come ci si possa difendere da un eventuale attacco nucleare sovietico: il compito della rete ARPANET è quello di collegare i principali centri di comando degli Stati Uniti, in modo tale che se uno di questi centri dovesse essere colpito, la rete non venga distrutta, ma possa comunque continuare a funzionare (trasmettere messaggi) utilizzando gli altri nodi. L'aspetto innovativo era costituito proprio dalla possibilità di definire una rete priva di una "gerarchia trasmissiva", in cui potessero essere presenti percorsi differenti per connettere due nodi e in cui la ricerca del percorso migliore (più breve o più efficiente) per trasmettere dati da un nodo della rete ad un altro, potesse essere calcolato dinamicamente sulla base dello stato dei nodi della rete stessa.

Inizialmente esistevano solo pochi nodi sperimentali di questa rete militare, ma ben presto il progetto, che rivestiva anche un grande interesse dal punto di vista teorico oltre che applicativo, si è allargato a numerose università e centri di ricerca degli Stati Uniti, così che dai pochi nodi iniziali si è passati nel giro di pochi anni alla rete informatica più grande del mondo: 4 nodi nel 1969, 15 nel 1971, più di 1.000 nel 1984, più di 10.000 nel 1987, più di 100.000 nel 1989, ... una crescita irrefrenabile! Col passare del tempo il progetto ha perso il suo significato militare e strategico (nel frattempo è finita anche la guerra fredda) ed ha acquistato un grandissimo interesse pratico, tanto che nel 1990 il progetto ARPANET ha cessato di esistere. Oggi i nodi collegati alla rete Internet sono milioni e sono sparsi in tutto il mondo.

Naturalmente è necessario che esista una forma di coordinamento tra gli utenti della rete per far sì che un sistema talmente vasto funzioni in modo congruente.

Sebbene il protocollo di comunicazione adottato per la trasmissione dei dati (il TCP/IP – *Transmission Control Protocol/Internet Protocol*) non stabilisca delle gerarchie tra i nodi della rete, è necessario un certo coordinamento per definire la modalità con cui i nodi possono continuare ad aggiungersi alla rete; al tempo stesso, per non ostacolare la crescita del sistema e per evitare che si creino isole di incompatibilità con il resto della rete, è necessario da un lato delegare e distribuire le responsabilità di gestione dei segmenti di rete e, dall'altro, garantire un coordinamento tecnico anche per la definizione di standard per lo sviluppo di nuove soluzioni tecnologiche.

Organismi di coordinamento della rete Internet

Per questo sono stati creati negli anni diversi organismi internazionali, coordinati fra loro, finalizzati a definire gli standard tecnici ed anche a delegare la gestione degli indirizzi di rete assegnati ad ogni macchina collegata alla rete Internet. Tra questi organismi possiamo ricordare la *Internet Society*, il *World Wide Web Consortium* (W3C), la *Internet Corporation for Assigned Names and Numbers* (ICANN), la *Internet Assigned Number Authority* (IANA), il RIPE (*Réseaux IP Européens*) e, in Italia, NIC-IT (*Italian Network Information Center*) e il GARR (*Gruppo di Armonizzazione delle Reti di Ricerca*). Le linee guida di sviluppo tecnico vengono definite e proposte alla comunità degli utenti della rete dalla *Internet Engineering task Force* (IETF) che periodicamente pubblica il risultato dei propri lavori nelle note conosciute come RFC (*Request for Comments*) disponibili al pubblico ad esempio sul sito Internet http://www.ietf.org/rfc.html.

Un contributo determinante all'introduzione e alla diffusione di Internet in Italia è stato offerto dall'associazione "*i2u*" (*Italian Unix Users*), l'associazione italiana degli utenti Unix, di cui facevano parte le principali aziende di informatica italiane (prima fra tutte Olivetti) e numerosi centri di ricerca pubblici ed istituti universitari italiani.

5.2 IP address e routing

A livello fisico la rete Internet è costituita da un insieme eterogeneo di computer collegati fra loro mediante le connessioni più diverse: convivono sulla Rete macchine Unix, personal computer e server in ambiente Microsoft Windows, supercomputer con sistemi operativi proprietari, apparati di rete dedicati alla gestione del traffico con sistemi operativi specializzati nell'esecuzione di tali operazioni e moltissimi altri sistemi e macchinari. Le connessioni di rete sono basate su linee seriali e telefoniche a bassa velocità, segmenti di collegamento su fibra ottica o su cavi di rete Ethernet, connessioni *frame relay* a banda larga, connessioni satellitari, wireless, cellulari GSM/GPRS/UMTS e molte altre ancora. L'unico aspetto che unifica questo minestrone di tecnologie è il protocollo di comunicazione TCP/IP che, a livelli e con implementazioni diverse, consente a questi apparati di integrarsi in modo del tutto trasparente per l'utente: fortunatamente non è necessario conoscere questi aspetti tecnologici per poter utilizzare la rete Internet. Nelle pagine seguenti faremo solo qualche accenno agli aspetti che riguardano più da vicino un utente di un sistema Unix.

Un insieme eterogeneo di connessioni e sistemi informatici

Indirizzamento dei nodi della rete

Ad ogni singola macchina collegata alla rete Internet viene assegnata dall'organismo di coordinamento regionale un indirizzo di rete univoco, chiamato *Internet number* o anche *IP address*. È costituito da una quaterna di numeri compresi tra 0 e 255. Ad esempio un numero valido può essere 193.204.165.209.

Per consentire una gestione "delegata" del processo di assegnazione degli indirizzi a tutti gli enti, le aziende e gli organismi che ne fanno richiesta, l'insieme di tutti i possibili indirizzi IP (sono più di quttro miliardi!) è stato suddiviso in *classi*; una classe o parte di essa viene assegnata in gestione ad ogni organismo che ne fa richiesta. A seconda delle esigenze dell'ente che richiede di entrare in possesso di

una classe di indirizzi IP, possono essere assegnate classi (o sottoclassi) che contengono pochi indirizzi o anche molte migliaia. Le classi di tipo "A" contengono circa 16 milioni di indirizzi; le classi di tipo "B" ne contengono circa 65.000 ciascuna ed infine le classi di tipo "C" contengono ognuna 254 indirizzi. Tra queste classi ne sono state individuate alcune che sono costituite da indirizzi riservati alle reti TCP/IP private, che non possono quindi essere collegate direttamente alla rete pubblica Internet perché altrimenti entrerebbero in conflitto con gli stessi indirizzi utilizzati da altre reti private. Per collegare una rete privata ad Internet è necessario quindi predisporre un gateway che consenta di veicolare i pacchetti di dati dalla rete privata verso l'esterno e viceversa. Le reti private sono generalmente identificate da indirizzi del tipo 10.$x.y.z$, per reti di grandi dimensioni, o 192.168.$x.y$, per reti di dimensioni più modeste. L'indirizzo IP 127.0.0.1 è un indirizzo riservato e viene assegnato ad ogni macchina che supporta il protocollo TCP/IP per indicare l'indirizzo locale (*localhost*).

Classi e sottoclassi di indirizzi IP

Indirizzi riservati

Ai computer facenti parte di una stessa rete locale (la rete interna di un'azienda, di un ufficio o di un dipartimento universitario), vengono assegnati indirizzi selezionati nella classe IP dell'organizzazione. Tali indirizzi saranno tutti piuttosto simili, a meno di almeno uno dei quattro numeri della quaterna che forma un *Internet number*. Ad esempio, se è stata assegnata la classe C 192.106.24.x, allora tutti gli *host* di quella rete avranno indirizzi che iniziano con 192.106.24 e terminano con un ultimo numero che è differente per ogni macchina (192.106.24.1, 192.106.24.2, fino a 192.106.24.254; il numero 192.106.24.0 rappresenta l'intera rete). Ad una stessa macchina possono essere attribuiti, per ragioni tecniche particolari, anche due o più indirizzi IP.

Ogni rete è collegata al resto della rete Internet attraverso un *gateway* costituito da un apparato denominato *router*. Queste macchine hanno almeno due interfacce di rete: ad una viene collegata la rete "interna", mentre l'altra viene utilizzata per connettersi alla rete "esterna" e da questa al resto di Internet. Ogni router ha quindi almeno due indirizzi IP: uno per il collegamento con la rete esterna e l'altro della stessa classe della rete IP interna.

Routing sulla rete

Quando una macchina intende comunicare con un altro computer identificato da un certo indirizzo IP, verifica innanzi tutto se l'indirizzo di destinazione appartiene o meno alla propria rete; se gli indirizzi IP delle due macchine (mittente e destinatario) fanno parte della stessa classe, allora la comunicazione avviene in modo diretto. Altrimenti la macchina "mittente", contatta il *gateway* della propria rete che provvederà ad inoltrare il pacchetto di dati verso la rete esterna fino a raggiungere il destinatario, sfruttando i *gateway* delle reti limitrofe.

Domain Name System e gerarchia dei nomi di dominio

L'utilizzo degli indirizzi IP numerici è piuttosto scomodo, dal momento che i numeri di per sé non dicono nulla dell'identità dell'ente o dell'organizzazione della cui rete fa parte un certo computer. È per questo che dal 1984 è stato introdotto sulla rete Internet un sistema denominato DNS (*Domain Name System*), che consente di associare un "nome di dominio" ad ogni rete e ad ogni host collegato ad Internet. Si tratta di una specie di grande "elenco telefonico" che è in grado di associare ad ogni *Internet number* un nome facilmente identificabile. Anche in questo caso è stato adottato un meccanismo in grado di poter delegare la gestione dell'assegnazione dei nomi ad un insieme di organismi di coordinamento territoriali. Dunque l'insieme dei nomi che è possibile assegnare ai nodi della rete Internet è stato suddiviso in domini su vari livelli: in cima a questa gerarchia "ad albero" ci sono i cosiddetti *Top Level Domains* (TLDs), che possono essere di tipo geografico o non geografico: si distinguono così i "*country code Top Level Domains*" (ccTLDs), come "it" per l'Italia, "es" per la Spagna, "uk" per il Regno Unito, ..., ed i TLD che non fanno riferimento a nessuno specifico Paese: "com" usato da molte aziende commerciali, "org" per le organizzazioni internazionali, "net" per gli enti di gestione della rete, ed altri ancora.

Ognuno di questi *Top Level Domain* viene gestito da un organismo nazionale o sovranazionale, che controlla il processo di registrazione ed assegnazione di un nome di dominio di "secondo livello" da collocare sotto alla gerarchia del proprio TLD. Sono stati così registrati moltissimi (milioni) domini di secondo livello che identificano bene o male tutte le organizzazioni, gli enti, le aziende, le istituzioni che in qualche modo sono connesse alla rete Internet (magari anche solo per la pubblicazione di un sito web). Ad esempio "uniroma3.it" è il dominio Internet di secondo livello assegnato all'Università Roma Tre, mentre "ibm.it" ed "ibm.com" sono due domini di secondo livello assegnati rispettivamente alla filiale italiana dell'IBM e alla IBM americana.

Nell'ambito del proprio dominio ogni organizzazione è responsabile della assegnazione e della gestione di eventuali sottodomini e degli *hostname* assegnati ad ogni singola macchina. Così organizzazioni più piccole possono decidere di mantenere un solo livello gerarchico nei propri nomi di dominio, mentre altre strutture più articolate possono suddividere il contesto generale in ulteriori domini di livello inferiore; ad esempio il dominio di secondo livello dell'Università Roma Tre contiene altri domini di terzo livello per ogni dipartimento della stessa Università: il dipartimento di Matematica ha il dominio "mat.uniroma3.it", quello di Fisica ha il dominio "fis.uniroma3.it", e così via.

Hostname e fully qualified domain name

All'interno di ogni dominio vengono assegnati gli *hostname* alle singole macchine collegate alla rete: archimede.mat.uniroma3.it, venere.mat.uniroma1.it, sono rispettivamente gli *hostname* delle macchine archimede e venere del Dipartimento di Matematica dell'Università Roma Tre e dell'Università di Roma "La Sapienza". Quando un *hostname* è composto anche dai nomi di dominio fino al TLD di appartenenza, allora quel nome è un *fully qualified domain name*.

È chiaro quindi che per tentare di decifrare il nome dell'organizzazione che "si cela" dietro ad un certo indirizzo Internet, dovremo leggerlo da destra verso sinistra: in questo modo viene letto per primo il dominio di più alto livello, per poi specificare via, via i vari sottodomini a cui tale macchina appartiene, come in un gioco di scatole cinesi.

Risoluzione di indirizzi, name server autoritativo

Quando una macchina deve comunicare con un'altra di cui conosce il *fully qualified domain name*, ma non l'indirizzo IP, deve tradurre quel nome in un indirizzo (in gergo si dice che deve "risolvere" quel nome) prima di potergli spedire il pacchetto di dati. Per far questo deve contattare un DNS server che, collegato con tutti gli altri DNS server presenti sulla rete Internet, è in grado di tradurre il nome in un indirizzo, oppure di affermare con certezza che quel nome non corrisponde a nessun indirizzo presente in rete. Ogni dominio Internet fa riferimento ad uno o più DNS server "autoritativi" per tale dominio: ogni volta che viene modificato l'indirizzo di una macchina all'interno di un dominio, deve essere aggiornato il DNS server autoritativo per quella zona, in modo tale da poter gestire correttamente la risoluzione dei nomi e degli indirizzi.

5.3 Risoluzione di nomi e di indirizzi

In ambiente Unix sono disponibili moltissimi comandi che consentono di verificare la configurazione della rete e di acquisire informazioni circa i nomi e gli indirizzi assegnati a sistemi facenti parte della nostra o di altre reti connesse ad Internet. Di seguito vedremo soltanto alcuni fra i comandi principali.

Visualizzazione e assegnazione di un hostname

Il primo comando di questa breve rassegna è `hostname` che, naturalmente, consente di visualizzare l'*hostname* del nostro computer. Lo stesso comando può essere utilizzato anche dall'utente *root*, con l'opzione "`-s`", per assegnare un nome al sistema.

Per risolvere un indirizzo traducendo un nome in un *IP address* o viceversa, si può utilizzare il comando `nslookup` che consente di interrogare i DNS server. Questo comando può essere utilizzato in modalità diretta (specificando opzioni e parametri sulla linea di comando) o interattiva, entrando in un ambiente in cui è possibile effettuare diverse richieste. Ogni interrogazione consiste nello specificare il tipo di informazione che si desidera ricevere e l'indirizzo che si vuole risolvere. Di seguito riportiamo una sessione di lavoro interattiva con `nslookup`.

Risoluzioni di indirizzi, nslookup

```
$ nslookup
> set type=ns
> www.mat.uniroma3.it.
Server:         192.106.1.1
Address:        192.106.1.1#53

Non-authoritative answer:
www.mat.uniroma3.it     canonical name = web2.mat.uniroma3.it.

Authoritative answers can be found from:
mat.uniroma3.it
        origin = dns.uniroma3.it
        mail addr = root.dns.uniroma3.it
        serial = 2005041400
        refresh = 86400
        retry = 1800
        expire = 2592000
        minimum = 172800
> server dns.uniroma3.it
Default server: dns.uniroma3.it
Address: 193.205.139.10#53
> set type=a
> www.mat.uniroma3.it.
Server:         dns.uniroma3.it
Address:        193.205.139.10#53

www.mat.uniroma3.it     canonical name = web2.mat.uniroma3.it.
Name:   web2.mat.uniroma3.it
Address: 193.204.165.209
> exit
$ _
```

Nell'esempio il nostro scopo è quello di stabilire quale sia l'indirizzo IP assegnato alla macchina identificata dal nome www.mat.uniroma3.it. Per prima cosa quindi, con il comando "`set type=ns`" abbiamo selezionato una interrogazione per conoscere l'indirizzo del DNS server autoritativo per l'*hostname* di nostro interesse; quindi abbiamo interrogato il nostro DNS server di default a proposito dell'indirizzo www.mat.uniroma3.it. Questo ci ha risposto che la risposta che ci stava fornendo non è autoritativa e che il DNS server da interrogare è dns.uniroma3.it. Abbiamo quindi cambiato DNS server con il comando "`server dns.uniroma3.it`"; quindi abbiamo cambiato il tipo di *query* con il comando "`set type=a`" che ci permette di conoscere l'indirizzo (*address*) associato ad un certo nome (e viceversa) ed abbiamo interrogato il nuovo DNS server a proposito dell'indirizzo www.mat.uniroma3.it. Questo finalmente ci ha risposto dicendo che l'indirizzo IP associato a quel nome è 193.204.165.209.

È possibile effettuare una interrogazione del DNS server anche per la risoluzione inversa: ad esempio supponiamo di voler conoscere il nome associato all'indirizzo 192.106.24.10; possiamo utilizzare il seguente comando, questa volta utilizzando il programma `nslookup` in modalità non interattiva:

```
$ nslookup 192.106.24.10
Server:         192.106.1.1
Address:        192.106.1.1#53

10.24.106.192.in-addr.arpa      name = sole.isinet.it.
$ _
```

Informazioni sui domini registrati, whois

Possiamo conoscere più da vicino le organizzazioni a cui sono stati assegnati i nomi di dominio Internet interrogando il cosiddetto database WHOIS, ossia il grande archivio in cui sono memorizzate le informazioni relative a tutti i domini registrati sulla rete Internet. In questo caso il comando da utilizzare è `whois`. La sintassi del comando è molto semplice, ma può variare da un sistema all'altro: in sostanza si tratta di indicare il nome del dominio di cui vogliamo ottenere informazioni ed eventualmente il nome di uno specifico WHOIS server. Vediamo un esempio:

```
$ whois uniroma3.it
domain:      uniroma3.it
org:         Terza Universita' degli Studi di Roma
descr:       Italian 2-th level domain
descr:       of the Third University of Rome
admin-c:     MRC25-ITNIC
tech-c:      PC1670-ITNIC
postmaster:  PC1670-ITNIC
zone-c:      PC1670-ITNIC
nserver:     193.205.139.10 dns.uniroma3.it
nserver:     193.204.5.4 decsrv.caspur.it
nserver:     193.205.245.66 dns3.nic.it
mnt-by:      GARR-MNT
created:     before 19960129
expire:      20060129
source:      IT-NIC

person:      Maria Rosaria Cagnazzo
address:     Rettorato Terza universita' di Roma
address:     Via Ostiense, 159
address:     I-00154 Roma
address:     Italy
nic-hdl:     MRC25-ITNIC
source:      IT-NIC

person:      Paolo Cursi
address:     Rettorato Terza universita' di Roma
address:     Via Ostiense, 159
address:     I-00154 Roma
address:     Italy
nic-hdl:     PC1670-ITNIC
source:      IT-NIC
$ _
```

Tra le molte informazioni visualizzate ce ne sono alcune particolarmente interessanti, quali gli indirizzi dei DNS server primari per il dominio e i riferimenti delle persone che si occupano della gestione della rete (*tech-c*, *zone-c*) e della posta elettronica (*postmaster*) per il dominio stesso. Il nominiativo identificato dall'etichetta *admin-c* è il cosiddetto "contatto amministrativo", ossia il nome del rappresentante dell'ente a cui è stato intestato il dominio.

Verifica dello stato di efficienza della rete, ping, traceroute

Per verificare lo stato della connessione di rete nel tratto che va dal nostro sistema ad un determinato altro computer, si possono utilizzare i comandi `ping` e `traceroute`. Ultimamente, con l'introduzione di apparati di controllo e filtraggio delle reti connesse ad Internet l'attendibilità dell'output di questi strumenti è un po' diminuita, ma consentono comunque di effettuare un test preliminare sullo stato di efficienza della rete.

Il comando `ping` consente di inviare (ripetutamente o una sola volta) un pacchetto di dati alla macchina di destinazione, attendendo poi da questa un *feedback* circa l'avvenuta ricezione del pacchetto. Il comando `traceroute` consente invece di tracciare il percorso (*routing*) dei pacchetti di dati che transitano dalla nostra macchina fino a quella di destinazione. Vediamo un esempio:

```
$ ping www.mat.uniroma3.it
PING web2.mat.uniroma3.it (193.204.165.209): 56 data bytes
64 bytes from 193.204.165.209: icmp_seq=0 ttl=52 time=13.765 ms
64 bytes from 193.204.165.209: icmp_seq=1 ttl=52 time=12.853 ms
64 bytes from 193.204.165.209: icmp_seq=2 ttl=52 time=13.416 ms
^C
--- web2.mat.uniroma3.it ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max/stddev = 12.853/13.345/13.765/0.376 ms

$ traceroute www.mat.uniroma3.it
traceroute to www.mat.uniroma3.it (193.204.165.209), 64 hops max, 40 byte
packets
 1  192.168.1.1 (192.168.1.1)  2.546 ms  1.204 ms  0.965 ms
 2  151.6.146.65 (151.6.146.65)  9.410 ms  11.336 ms  11.306 ms
 3  rmcc-b01-ge2-0.31.wind.it (151.6.73.65)  9.981 ms rmcc-b01-ge10-0.41.wi
nd.it (151.6.73.81)  11.631 ms rmcc-b02-ge2-0.31.wind.it (151.6.73.66)
13.102 ms
 4  151.6.4.53 (151.6.4.53)  36.969 ms  9.861 ms 151.6.5.41 (151.6.5.41)
9.993 ms
 5  fici-b01-rmid-t01-po01.wind.it (151.6.1.78)  11.496 ms 151.6.5.58 (151.
 6.5.58)  12.779 ms  10.602 ms
 6  garr-nap.namex.it (193.201.28.15)  11.399 ms  12.267 ms  10.618 ms
 7  rt-rm2-rt-rm1-1.rm1.garr.net (193.206.134.197)  11.799 ms  11.040 ms
11.879 ms
 8  193.206.131.146 (193.206.131.146)  12.805 ms  12.513 ms  11.900 ms
 9  * * *
10  natamm.cab.uniroma3.it (193.204.167.85)  31.213 ms  13.786 ms 14.578 ms
11  168ext7.cab.uniroma3.it (193.204.163.2)  14.290 ms  14.250 ms 13.095 ms
12  168ext7.cab.uniroma3.it (193.204.163.2)  16.111 ms  19.129 ms 34.813 ms
13  www.mat.uniroma3.it (193.204.165.209)  14.806 ms !<10> 15.855 ms !<10>
14.751 ms !<10>
$ _
```

L'output del comando `ping` ci dice che la connessione funziona piuttosto bene, dal momento che nessuno dei pacchetti di dati trasmessi è andato perso. Per interrompere il comando `ping` sono stati battuti i tasti Ctrl-c. L'output di `traceroute` ci permette di scoprire che per comunicare i due computer devono passare attraverso 12 nodi intermedi: il primo (identificato dall'indirizzo di rete privata "192.168.1.1")

è l'indirizzo del *gateway* della rete di partenza, gli altri sono nodi intermedi, mentre l'ultimo (il tredicesimo nodo dell'elenco) è la macchina di destinazione.

5.4 Sessioni di lavoro su server remoti

Una delle funzionalità più interessanti offerte da un server Unix è la possibilità di aprire delle sessioni di lavoro con la *shell* da postazioni remote, ossia sfruttando come un terminale un computer collocato anche a grande distanza dal server Unix su cui intendiamo lavorare; il collegamento tra la postazione di lavoro ed il server Unix remoto è una connessione di rete TCP/IP e non un collegamento "diretto" mediante una linea seriale, come avviene con i tradizionali terminali alfanumerici. Per attivare una sessione di lavoro da remoto si sfruttano dei sistemi di tipo *client/server* in cui la macchina Unix su cui vogliamo operare gioca il ruolo di server per la connessione effettuata dal *client* costituito dalla postazione di lavoro su cui ci troviamo; questa può essere un personal computer in ambiente Windows, un Macintosh, un'altra macchina Unix o qualunque altro genere di computer e sistema operativo, purché dotato del software *client* necessario per gestire la sessione di lavoro remota in collegamento via rete con il server Unix.

In sostanza ciò che si ottiene aprendo una sessione di lavoro remota non è altro che una *shell* di comandi Unix che viene eseguita sul server remoto, ma con cui è possibile interagire utilizzando i dispositivi di input/output della postazione di lavoro locale (essenzialmente il video con cui visualizzare l'output dei processi eseguiti sul server remoto e la tastiera per impartire i comandi). In questo modo, anche per eseguire programmi piuttosto impegnativi che richiedono notevoli risorse di calcolo, non è necessario disporre di una postazione di lavoro molto potente, dal momento che il programma che intendiamo utilizzare sarà eseguito sul server remoto sfruttando le sue risorse di memoria e di CPU.

Sessioni di lavoro remote, telnet

Storicamente uno dei primi sistemi *client/server* per attivare sessioni di lavoro remote è Telnet. Si tratta di un sistema costituito da due programmi, il server o "demone Telnet" (`telnetd`) che consentiva di accettare e gestire le connessioni dalle postazioni di lavoro remote, e il client Telnet, implementato praticamente su ogni sistema operativo (anche Windows ha un suo client Telnet). In ambiente Unix il client può essere richiamato con il comando `telnet`, specificando poi sulla riga di comando l'indirizzo o l'*hostname* del server con cui si intende stabilire la connessione.

```
marco@aquilante ~$ telnet venere.mat.uniroma1.it
Trying 141.108.5.85...
Connected to venere.mat.uniroma1.it.
Escape character is '^]'.

SunOS UNIX (venere)
login: liverani
Password:

Last login: Mon Aug 29 11:22:11 from woodstock.mat.uniroma3.it
SunOS Release 4.1.3_U1 (GENERIC) #1: Wed Oct 13 17:48:35 PDT 1993
Venere (Sun4/60 del Dipartimento di Matematica)

liverani@venere ~> who
liverani   pts/1    Sep  4 11:28    (aquilante.isinet.it)
liverani@venere ~> _
```

Al momento del login vengono visualizzati alcuni messaggi che ci informano sull'ultima data di connessione e sulle caratteristiche del server su cui abbiamo aperto la sessione di lavoro. Ho utilizzato un *prompt* più sofisticato per evidenziare che il comando `telnet` è stato eseguito sulla postazione locale denominata "aquilante", mentre la sessione remota e l'esecuzione del comando `who` sono effettuati sul server denominato "venere"; anche lo *username* dell'utente è cambiato: su "aquilante" l'utente è identificato dallo *username* "marco", mentre su "venere" lo *username* è "liverani". Per scollegarsi dal sistema e terminare la sessione di lavoro remota si utilizzano i consueti comandi `exit` o `logout`.

Il messaggio "Escape character is..." indica che, in questo esempio, la sequenza di tasti Ctrl-] ci permette di rientrare nel sistema locale per chiudere la connessione, sospenderla, ecc. Premendo Ctrl-] ci viene presentato il *prompt* "`telnet>`" del programma `telnet`; ad esempio per chiudere la sessione a questo punto possiamo usare il comando `close`.

Il sistema Telnet negli ultimi anni è andato in disuso, venendo rapidamente sostituito da altri sistemi più sofisticati. Su molti server Unix il demone `telnetd` non è attivo e spesso non è neanche installato. Il motivo che ha fatto cadere in disgrazia un sistema che è stato molto utilizzato fino a qualche anno fa è costituito dal fatto che la sessione di lavoro in rete effettuata con il protocollo Telnet avviene "in chiaro": è abbastanza facile, quindi, utilizzando appositi strumenti di monitoraggio del traffico di rete (ad esempio `tcpdump` e `snoop`) intercettare i pacchetti scambiati tra il client ed il server Telnet durante la sessione di lavoro; in questo modo sono state intercettate moltissime *password* di utenti del tutto ignari del pericolo che stavano correndo.

La soluzione al problema non ha tardato ad arrivare ed è costituita da un nuovo protocollo di comunicazione che consente di stabilire delle sessioni di lavoro remote in tutta sicurezza, dal momento che ogni pacchetto di dati scambiato tra il client ed il server viene cifrato utilizzando algoritmi crittografici molto potenti e robusti.

Sessioni sicure, ssh/ssh2

Il metodo più diffuso oggi è quello offerto dal sistema SSH2 (*Secure Shell* versione 2): anche in questo caso il sistema si compone di due programmi, un client ed un server, che dialogano fra loro utilizzando un protocollo di comunicazione (SSH2); il protocollo prevede che entrambe le controparti eseguano la cifratura delle informazioni scambiate con algoritmi di crittografia asimmetrica. In questo modo è impossibile decifrare il contenuto dei pacchetti di dati trasmessi durante la sessione di lavoro. Per aprire una sessione di lavoro sicura su un server su cui è attivo il demone `sshd` (la componente server del sistema) si può utilizzare il comando `ssh2`[1] che attiva il client per la connessione al server remoto. Anche sui sistemi operativi non Unix sono disponibili client SSH2 con cui è possibile aprire una sessione di lavoro sicura su un server Unix remoto; ad esempio per l'ambiente Windows si possono facilmente reperire in rete ed installare i programmi WinSSH e PuTTY.

Sulla linea di comando di `ssh2` si deve specificare anche l'indirizzo o l'*hostname* del server a cui intendiamo collegarci e lo *username* dell'account che vogliamo utilizzare su tale sistema, se è differente da quello utilizzato sulla macchina locale, usando la sintassi "*username*@*hostname*":

```
marco@aquilante ~$ ssh2 liverani@venere.mat.uniroma1.it

The authenticity of host 'venere.mat.uniroma1.it (141.108.5.85)'
can't be established.
DSA key fingerprint is 91:2f:65:b6:58:2e:8b:96:97:63:61:f4:89:0f:ad.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'venere.mat.uniroma1.it,141.108.5.85'
```

[1] Su alcuni sistemi questo comando non esiste: si deve utilizzare il comando "`ssh -2`".

```
(DSA) to the list of known hosts.

liverani@venere.mat.uniroma1.it's password:
Last login: Mon Aug 29 2005 16:25:49
SunOS Release 4.1.3_U1 (GENERIC) #1: Wed Oct 13 17:48:35 PDT 1993
Venere (Sun4/60 del Dipartimento di Matematica)

You have new mail.
liverani@venere ~> _
```

I messaggi relativi all'autenticità del server visualizzati all'inizio della connessione indicano che è la prima volta che l'utente si connette via SSH2 a quell'indirizzo e dunque la chiave di identificazione del server viene aggiunta in un file presente nella *home directory* locale dell'utente (`~/.ssh/known_hosts`) in modo tale che nelle connessioni successive non venga più visualizzato questo messaggio di allarme.

La sessione di lavoro può essere chiusa digitando i soliti comandi `exit` o `logout`.

Il sistema SSH2 consente di sostituire il tradizionale processo di autenticazione dell'utente basato sulla coppia *username/password*, con un meccanismo più sofisticato basato sul concetto di chiave publica e chiave privata che è alla base degli algoritmi di crittografia asimmetrica (come RSA e DSA, utilizzati da SSH2).

Autenticazione mediante chiave pubblica e privata, ssh-keygen

Se un utente si collega frequentemente in SSH2 da una stessa postazione di lavoro ad un insieme di *host* ben noti, allora può trovare più comodo usare le chiavi asimmetriche, anziché dover digitare ogni volta la propria *password*. Per predisporre le chiavi da usare per l'identificazione dell'utente si deve utilizzare il programma `ssh-keygen`, che genererà due file, uno contenente la chiave privata dell'utente (`id_rsa`) e l'altro la sua chiave pubblica (`id_rsa.pub`). Durante il processo di generazione delle chiavi viene richiesta l'immissione di una *password* facoltativa: siccome vogliamo effettuare il *login* sulla macchina remota senza dover digitare una *password* per sbloccare la chiave, è necessario non digitare nessuna *password*; questo non indebolisce in modo significativo il meccanismo di autenticazione. Il programma `ssh-keygen` deve essere eseguito sulla macchina da cui vogliamo stabilire la connessione remota (il client SSH2). Nell'esempio che segue chiediamo di generare una coppia di chiavi RSA (opzione "`-t rsa`"):

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/marco/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/marco/.ssh/id_rsa.
Your public key has been saved in /home/marco/.ssh/id_rsa.pub.
The key fingerprint is:
53:e4:ac:41:9b:17:cc:4b:22:3d:15:6a:59:e4:d6:a4 marco@woodstock
$ ls .ssh/
id_rsa         id_rsa.pub     known_hosts
$ _
```

Per completare la configurazione del meccanismo di autenticazione con chiavi RSA, si deve copiare la chiave pubblica (il contenuto del file "`~/.ssh/id_rsa.pub`") in un file denominato "`~/.ssh/authorized_keys`" sui server remoti a cui intendiamo collegarci. Il file "`authorized_keys`" può contenere anche più di una chiave pubblica (una per ogni postazione di lavoro da cui avviene la connessione SSH2), purché queste siano contenute ognuna su una sola riga di testo del file. Su altri sistemi la chiave pubblica deve essere copiata in un file a sé stante (es.: "`~/.ssh/chiave.pub`") e

nel file "`~/.ssh/authorization`" deve essere riportato un riferimento a quel file in una riga scritta con la seguente sintassi "`key` *filename*" (es.: "`key chiave.pub`").

A questo punto la connessione sicura può essere effettuata senza la necessità di digitare nessuna *password*:

```
marco@aquilante ~$ ssh2 liverani@venere.mat.uniroma1.it
Last login: Sun Sep 4 2005 12:46:17
SunOS Release 4.1.3_U1 (GENERIC) #1: Wed Oct 13 17:48:35 PDT 1993
Venere (Sun4/60 del Dipartimento di Matematica)

You have mail.
liverani@venere ~> _
```

Tunnel SSH2 per sessioni X Window

Con SSH2 è possibile aprire una sessione di lavoro remota su un server e creare un *tunnel* cifrato sul canale di comunicazione stabilito tra il client ed il server, entro cui far passare anche i messaggi scambiati nell'ambito della gestione dell'output grafico in ambiente X Window. Questo garantisce che anche l'esecuzione di applicazioni che richiedono una interazione con l'utente attraverso un server grafico X11, avvenga in tutta sicurezza impedendo ad altri utenti connessi in rete di interpretare i messaggi scambiati tra le applicazioni eseguite sul server e la *workstation* grafica su cui opera l'utente. Per aprire il *tunnel SSH* per la sessione X Window è necessario che il server sia stato configurato per sfruttare questa caratteristica. In tal caso il client `ssh2` può essere eseguito con l'opzione "`-X`": sul server viene intercettato l'output grafico diretto al display `localhost:10.0` che viene invece cifrato ed inviato al display della *workstation* remota.

5.5 Navigazione nel World Wide Web

Il *web* (o più esattamente il *World Wide Web* o WWW) oggi è sicuramente uno degli aspetti più appariscenti e utilizzati della rete Internet; è diventato molto rapidamente uno dei principali canali di promozione e diffusione delle informazioni. È il risultato di studi iniziati negli anni '80 da Tim Berners Lee[2] e da pochi altri ricercatori del CERN di Ginevra, che con questo nuovo potente strumento hanno reso accessibile a milioni di nuovi utenti "non tecnici" l'uso della rete Internet.

HTTP, HTML, URI e URL

Il web si basa fondamentalmente su su un protocollo di comunicazione chiamato HTTP (*HyperText Transfer Protocol*) utilizzato da appositi software che operano in modalità *client/server*: il server HTTP (o anche HTTP *daemon* – `httpd`) ed il client costituito da un *web browser*. I documenti, i file, le immagini vengono pubblicati sulla rete da un server web che permette di identificare ogni singolo file pubblicato con un indirizzo denominato URI (*Uniform Resource Identifier*) o URL (*Uniform Resource Locator*). Utilizzando un client HTTP (un *web browser*) si può visualizzare il file identificato da una determinato indirizzo URL pubblicato in rete da un web server. I singoli documenti ipertestuali pubblicati sul WWW sono spesso codificati con un linguaggio di marcatura del testo chiamato HTML (*HyperText Markup Language*), che consente di dare una struttura logica al documento, di collegarlo a risorse di tipo multimediale (come immagini o sequenze audio/video) e di costruire riferimenti "ipertestuali" che consentono di "navigare" da un documento ad un altro ad esso collegato e che probabilmente è di contenuto affine.

In ambiente Unix il web server più diffuso è il programma *open source* Apache HTTP Server. Se sul server è disponibile ed è attivo Apache, allora probabilmente la configurazione del server web consente agli utenti di pubblicare sul

[2]La home page di Tim Berners Lee è disponibile all'indirizzo http://www.w3.org/People/Berners-Lee/.

Server web Apache, home page personale

web una propria *home page* personale. Nella configurazione di default di Apache la *home page* di ogni utente di un sistema Unix è costituita dai file contenuti nella directory "`~/public_html`" presente nella *home directory* dell'utente; tale *home page* è accessibile sul web mediante un indirizzo URL di questo tipo: "`http://`*hostname*`/~`*username*`/`" (ad esempio `http://www.isinet.it/~marco/`).

Web browser, Mozilla, Netscape, Lynx

Per accedere alle risorse disponibili sul web in ambiente Unix sono disponibili numerosi programmi client. Il principale e più diffuso è il potente *web browser* grafico Mozilla, che oltre ad offrire la possibilità di navigare sul web, mette a disposizione dell'utente anche un *client* di posta elettronica ed un *news reader*. Per lanciare il *browser* da un `xterm` sotto X Window si deve semplicemente digitare il comando "`mozilla &`". In alternativa a Mozilla è possibile utilizzare il programma Netscape; anche in questo caso è sufficiente digitare il comando "`netscape &`".

Per gli utenti che non hanno la possibilità di accedere ad un terminale grafico in ambiente X Window è disponibile un *web browser* che funziona in modalità testuale su qualsiasi terminale alfanumerico. Si chiama Lynx e può essere lanciato utilizzando il comando "`lynx`".

Download di file, wget

Se, invece di "navigare" sul web in modalità interattiva si desidera semplicemente scaricare nella directory locale un file di cui si conosce l'indirizzo URL, allora può essere utile il comando `wget`. Specificando l'indirizzo del file come parametro sulla linea di comando, questo verrà trasferito usando il protocollo HTTP e salvato in locale:

```
$ wget http://www.isinet.it/~marco/unix/manuale-unix.pdf
--16:39:05--  http://www.isinet.it/%7Emarco/unix/manuale-unix.pdf
           => `manuale-unix.pdf'
Resolving www.isinet.it... done.
Connecting to www.isinet.it[192.106.24.10]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 453,502 [application/pdf]
100%[=============================>] 453,502  142.59K/s ETA 00:00
16:39:08 (142.59 KB/s) - `manuale-unix.pdf' saved [453502/453502]
$ _
```

Il comando `wget` può essere anche utilizzato con l'opzione "`-r`" (*recursive*) per scaricare in locale un sito web a partire da una pagina specificata attraverso il suo indirizzo URL: `wget` seguirà i collegamenti ipertestuali presenti in quella pagina web e nelle pagine ad essa collegate, fino a raggiungere la "distanza" specificata dall'utente con l'opzione "`-l`" (di *default* è 5) dalla pagina iniziale. I file vengono salvati sul *filesystem* locale e, se viene specificata l'opzione "`-k`" i link ipertestuali vengono tradotti in modo tale che il sito web possa essere esplorato anche localmente. Ad esempio: "`wget -r -k -l 2 http://www.w3.org`".

5.6 Scambio di file con sistemi remoti

File transfer protocol, ftp

FTP significa *File Transfer Protocol* e serve appunto per trasferire file da una macchina all'altra collegate fra loro in rete TCP/IP (dunque anche su Internet). Si tratta ancora una volta di un protocollo di comunicazione adottato da appositi programmi in una modalità *client/server*: è necessario che su una macchina server sia attivo un demone FTP (`ftpd`) in modo tale che l'utente si possa collegare utilizzando un *client* FTP per scambiare (prelevare o depositare) file con il server. In ambiente Unix il programma client può essere richiamato con il comando `ftp` (lo stesso comando è anche disponibile sul sistema operativo Microsoft Windows). Per stabilire il collegamento FTP con un server bisogna specificare il comando seguito dall'indirizzo o dall'*hostname* del server a cui ci si intende connettere.

Anche in questo caso, connettendosi con un computer remoto per prelevare o depositare file mediante FTP, viene richiesto all'utente di farsi riconoscere mediante il suo *username* e la sua *password*. In alcuni casi sono disponibili dei servizi FTP "anonimi", ossia che consentono il collegamento e lo scambio di file anche ad utenti privi di un account di accesso al sistema stesso. In questi casi come *username* bisogna usare la parola riservata "`anonymous`" (o anche "`ftp`") e, come *password*, il proprio indirizzo di posta elettronica. In questo modo è possibile accedere al server FTP prelevando o depositando file in un'area pubblica in cui chiunque può avere accesso. È questo uno dei principali canali di scambio di informazioni, guide, manuali e software di pubblico dominio, che arricchisce e rende per certi versi unica la comunità degli utenti Internet.

Durante una sessione FTP si possono fare poche cose: sostanzialmente spostarsi all'interno del *filesystem* della macchina remota, prelevare o depositare file. Per far questo si deve utilizzare un ristretto set di comandi; riportiamo di seguito solo i principali, rimandando alla documentazione del comando `ftp` ("`man ftp`") per una illustrazione più chiara ed esauriente di tutti gli altri.

Principali comandi FTP

`cd` ha lo stesso uso e significato dell'omonimo comando della *shell* Unix: serve per cambiare la directory corrente nel *filesystem* del server FTP remoto;

`lcd` è analogo al comando `cd`, ma consente di cambiare directory sul *filesystem* della postazione locale;

`pwd` serve per visualizzare il nome della directory corrente sul server remoto (*print working directory*);

`bin` serve per indicare che le operazioni di trasferimento file devono avvenire in modalità "binaria" (per trasferire programmi, file in formato compresso, immagini, ecc.);

`ascii` serve per indicare che le operazioni di trasferimento file devono avvenire in modalità "*plain text*" (per trasferire file di testo ASCII);

`get` serve per trasferire un file dal sistema remoto al computer locale; la sintassi è "`get` *fileremoto filelocale*", dove *fileremoto* indica il nome del file che si trova sul server FTP remoto, e *filelocale* (che può anche essere omesso) indica il nome da assegnare al file quando sarà salvato nel disco del sistema locale;

`put` svolge la funzione inversa del comando `get`, trasferendo sul sito remoto un file residente sul disco del sistema locale; la sintassi è "put *filelocale fileremoto*";

`mget` consente di prelevare con uno stesso comando più di un file; accetta il carattere "`*`" per specificare il nome dei file da trasferire;

`hash` durante il trasferimento di un file visualizza lo stato di avanzamento della trasmissione, rappresentando sullo schermo una specie di "*progress bar*";

`dir` elenca i file contenuti nella directory corrente;

`quit` termina la sessione di collegamento FTP.

Un esempio di una sessione FTP è il seguente, che riporta un collegamento con il server finlandese "ftp.funet.fi", uno dei più grandi server FTP europei:

```
$ ftp ftp.funet.fi
Connected to ftp.funet.fi.
Name (ftp.funet.fi:marco): anonymous
```

```
331-Welcome to the FUNET anonymous ftp archive
331-
331-THIS is a four processor SUN 450/4GB/2.5TB system
331-Please mail to problems@nic.funet.fi in case of problems
Password:
Remote system type is UNIX.
ftp> cd pub/doc/unix
250 OK. Current directory is /.m/pub/doc/unix
ftp> dir
227 Entering Passive Mode (193,166,3,2,123,145)
150 Accepted data connection
-rw-r--r--    1 8129     999       2158072 Jul 26  1993 doc.tar.gz
drwxr-xr-x   11 8129     999          8192 Aug 11  1999 misc
drwxr-xr-x   20 8129     999          8192 Aug 11  1999 ps1
drwxr-xr-x    2 8129     999          8192 Aug 11  1999 run
drwxr-xr-x   36 8129     999          8192 Aug 11  1999 usd
ftp> bin
200 TYPE is now 8-bit binary
ftp> hash
Hash mark printing on (1024 bytes/hash mark).
ftp> get doc.tar.gz
local: doc.tar.gz remote: doc.tar.gz
227 Entering Passive Mode (193,166,3,2,89,115)
150-Accepted data connection
####################################################
226-File successfully transferred
2158072 bytes received in 00:19 (107.13 KB/s)
ftp> quit
221-Goodbye. You uploaded 0 and downloaded 2108 kbytes.
$ _
```

Nell'esempio ci siamo collegati con il sistema entrando come utenti "anonimi", ci siamo spostati nella directory "`/pub/doc/unix`", contenente documentazione sul sistema operativo Unix ed abbiamo prelevato il file `doc.tar.gz` in modalità binaria. Come al solito sperimentare questi comandi di persona aiuterà a capire il funzionamento di FTP molto più di qualsiasi spiegazione.

Copia sicura di file, scp

Un altro strumento assai utile per trasferire un file da una macchina ad un'altra è costituito dal comando `scp` (*Secure Copy*), una componente del sistema SSH2. Questo comando è simile al comando `cp` utilizzato per copiare file all'interno del *filesystem*, ma a differenza di quest'ultimo, consente la copia del file tra due macchine distinte, cifrando con gli stessi algoritmi utilizzati da `ssh2` tutti i dati trasmessi durante il trasferimento. Per l'identificazione dell'utente sulla macchina remota si utilizza la coppia *username/password* oppure, se sono state create e configurate le chiavi RSA/DSA, l'autenticazione con chiavi asimmetriche.

Sulla linea di comando si deve specificare il nome del file da copiare e poi il nome del file di destinazione; il file "remoto" deve essere specificato con la sintassi "*username@hostname:filename*". Nel seguente esempio trasferiamo il file "`relazione.pdf`" dal computer locale a quello remoto e il file "`dijkstra.c`" dal server remoto alla macchina locale:

```
$ scp relazione.pdf liverani@aquilante.mat.uniroma3.it:relazfin.pdf
liverani@aquilante.mat.uniroma3.it's password:
relazione.pdf             100%    14329     11.0KB/s   00:00
$ scp liverani@aquilante.mat.uniroma3.it:src/dijkstra.c dijkstra.c
liverani@aquilante.mat.uniroma3.it's password:
```

```
src/dijkstra.c            100%      562      9.0KB/s   00:00
$ _
```

5.7 Le News Usenet

La posta elettronica è un sistema privato di comunicazione tra utenti. Teoricamente nessuno, oltre al mittente e al destinatario del messaggio, può leggere il contenuto del messaggio stesso. Spesso può essere utile far sì che un proprio messaggio su un determinato tema venga condiviso con molti altri utenti della Rete, anche sconosciuti. A questo scopo, su Internet sono stati creati dei gruppi di discussione a tema (*newsgroup*), che raccolgono ognuno gli articoli dei propri lettori su un determinato argomento. Chiunque può inviare un articolo ad un *newsgroup* che verrà letto da tutti gli utenti che seguono (leggono periodicamente) i nuovi messaggi di quel gruppo.

Le news e i newsgroup

Oggi i *newsgroup* attivi sono diverse migliaia, e il numero è destinato a crescere di giorno in giorno, vista la frequenza con cui vengono creati nuovi gruppi di discussione. Per orientarsi in questo mare di articoli, sarà bene chiarire che anche i *newsgroup* sono organizzati in una struttura ad albero in cui le varie aree (i rami dell'albero) sono chiamate *gerarchie*. Di seguito sono elencate alcune delle gerarchie principali (di primo livello nella struttura ad albero):

alt È la gerarchia "alternativa", in cui si ritrovano i newsgroup più impensabili ed in cui ogni argomento è trattato in modo assai particolare;

biz Gruppi di tipo "business";

comp È la gerarchia dei newsgroup riguardanti argomenti tecnici di *computer science*;

gnu Gruppi concernenti il software e le attività dello *GNU Project* e della *Free Software Foundation*;

sci Gerarchia dei gruppi riguardanti le discipline scientifiche;

soc Gruppi relativi agli aspetti sociali e culturali delle diverse nazioni del mondo.

Per gli utenti di lingua italiana è utile indicare l'esistenza dei gruppi della gerarchia "it" (di cui fa parte il *newsgroup* "it.comp.os.unix", dedicato al sistema operativo Unix) e del newsgroup "soc.culture.italian".

Alcuni gruppi sono "moderati", ossia esiste un utente (o un gruppo di utenti) che svolgono la funzione di moderatore della discussione e vagliano a priori l'inserimento o la cancellazione dei messaggi nel *newsgroup*.

Andremmo ben oltre le finalità di questa breve introduzione descrivendo le numerose "buone maniere" che è necessario adottare nella partecipazione ad un gruppo di discussione su Internet per non attirare su di noi le maledizioni degli altri utenti; ci limitiamo quindi a ricordare che non sempre sono gradite "firme" eccessivamente lunghe o "artistiche" alla fine dei messaggi e che è bene evitare di entrare in sterili polemiche (*flames*) con gli altri utenti del *newsgroup* o di intraprendere inutili guerre di religione (*holy war*) a favore di questo o quell'argomento. In poche parole è bene discutere pacatamente ed in un certo senso anche con umiltà, visto che i nostri interlocutori sono migliaia e sparsi in tutto il mondo (è difficile pensare di avere sempre ragione o di essere il migliore in una situazione di questo genere...).

Per accedere in lettura e in scrittura ai *newsgroup* delle News Usenet, si deve

Newsreader, tin, rtin, pine

utilizzare un *news reader*, come i programmi `tin` o `rtin` o lo stesso `pine` di cui abbiamo già parlato in precedenza. Tutti questi programmi permettono di selezionare (evidenziandoli con il cursore all'interno di una lista) il gruppo in cui "entrare" e, all'interno del gruppo, i nuovi messaggi che non abbiamo ancora letto. Con il *news reader* è possibile, come su un normale programma di posta elettronica, leggere i nuovi messaggi, ed inviare al *newsgroup* le nostre risposte o i nostri articoli originali. La gestione del *newsgroup* avviene su una macchina remota (*news server*) a cui il *news reader* si collega mediante il protocollo NNTP (*Network News Transport Protocol*). Ulteriori dettagli sulle modalità operative del programma utilizzato per leggere le *news* è possibile reperirle sulla documentazione del proprio sistema.

Figura 5.1: Una Digital Alphastation e un VAX in ambiente OpenVMS

Appendice A

Sintesi dei comandi principali

`acroread` Adobe Acrobat Reader in ambiente grafico X Window. Vedi anche `xpdf`.

`alias` Definisce un nome alternativo per un comando della *shell*.

`bash` Bourne-Again Shell è un interprete che esegue i comandi letti da tastiera o da un file. Vedi anche `csh`, `ksh`, `sh`,`tcsh`.

`bc` Calcolatrice per il terminale alfanumerico. Vedi anche `xcalc`.

`bg` Riprende in background l'esecuzione di un processo precedentemente sospeso battendo Ctrl-z. Come argomento del comando `bg` può essere specificato il *job number* (preceduto dal simbolo "`%`") o il *process ID* del processo da mandare in background. Ad esempio: "`bg %3`". Vedi anche `fg`.

`cal` Visualizza un calendario. Senza alcun parametro visualizza il calendario del mese corrente; si può anche specificare un particolare mese ed anno. Ad esempio: "`cal 7 1492`".

`calendar` Visualizza gli appuntamenti e le attività del giorno e di quello successivo.

`cat` Concatena i file specificati come argomento del comando e li invia allo *standard output*. Ad esempio: "`cat file1 file2 file3`". Vedi anche `less`, `more`.

`cd` Nella *shell* e in una sessione FTP cambia la directory corrente.

`chmod` Cambia i permessi di accesso ai file. Ad esempio: "`chmod 644 pippo`".

`clock` In ambiente grafico X Window visualizza una finestra con un orologio; `clock` è una utility dell'ambiente OpenWindow. Vedi anche `date`, `oclock`, `xclock`.

`compress` Comprime e decomprime un file. I file compressi con `compress` hanno estensione "`.Z`". Con il comando "`compress pippo`" si genera il file `pippo.Z`; per decomprimerlo, ottenendo il file originale, si usi il comando "`compress -d pippo.Z`". Vedi anche `gzip`, `uncompress`, `zip`.

`cp`	Copia i file. Il formato del comando è "`cp` [*opzioni*] *origine destinazione*", dove *origine* è il nome del (dei) file da copiare, mentre *destinazione* è il nome del file o della directory in cui copiare il (i) file *origine*. Ad esempio: "`cp pippo.* ./src`". Vedi anche `mv`, `scp`.
`csh`	C Shell. È un interprete che esegue i comandi letti da *standard input* o da un file. Vedi anche `bash`, `ksh`, `sh`, `tcsh`.
`date`	Visualizza o imposta la data e l'ora corrente.
`df`	Visualizza lo spazio ancora libero nel *filesystem*. Vedi anche `du`.
`display`	Visualizza immagini grafiche in una finestra sotto X Window. Vedi anche `xv`.
`du`	Visualizza l'occupazione (in in blocchi o in Kbyte) della directory specificata e delle sue sottodirectory. Vedi anche `df`.
`dvips`	Converte un documento dal formato DVI al PostScript. Ad esempio: "`dvips tesi.dvi -o tesi.ps`". Vedi anche `xdvi`.
`elm`	*Electronic Mail for Unix.* Programma interattivo per la lettura e la spedizione di messaggi di posta elettronica. Vedi anche `mail`, `pine`.
`emacs`	Editor per file di testo. Vedi anche `pico`, `vi`.
`exit`	Termina l'esecuzione della *shell*. Vedi anche `logout`.
`export`	Sotto Bourne Shell imposta il valore di una variabile di ambiente. Vedi anche `setenv`.
`fg`	Porta in *foreground* il processo specificato mediante il suo *process ID* o il suo *job number* (preceduto dal simbolo "`%`"). Ad esempio: "`fg 143`". Vedi anche `bg`.
`file`	Identifica il tipo di codifica delle informazioni contenute in un file.
`find`	Cerca un file nel *filesystem*.
`finger`	Programma per la visualizzazione di informazioni sugli utenti. Vedi anche `w`, `who`.
`ftp`	È il client per l'uso del protocollo di trasmissione file FTP (*File Transfer Protocol*); consente all'utente di trasferire file sulla rete dalla propria macchina ad un sito remoto e viceversa. Ad esempio: "`ftp ftp.cnr.it`". Vedi anche `scp`.
`ghostview`	Programma per la visualizzazione di un documento PostScript in una finestra grafica sotto X Window; utilizza il programma GhostScript (`gs`).
`gnuplot`	Programma per la visualizzazione di grafici di funzione in una o due variabili; consente anche di rappresentare in forma grafica dati numerici contenuti in file di testo.
`grep`	Cerca una stringa in un file di testo mediante un'espressione regolare.
`gs`	È il comando con cui si richiama il programma GhostScript, un interprete PostScript per visualizzare e stampare documenti in quel formato. Spesso è più comodo utilizzare GhostScript tramite `ghostview`, invece che utilizzare direttamente i comandi di `gs`.

`gunzip` Programma per decomprimere file compressi con `gzip`. Ad esempio: “`gunzip pippo.gz`”.

`gzip` Comprime e decomprime un file. I file compressi con `gzip` hanno estensione “`.gz`”. Con il comando “`gzip pippo`” si genera il file compresso `pippo.gz`; per riportarlo nel formato originale si usi il comando “`gzip -d pippo.gz`”. Vedi anche `compress`, `zip`.

`head` Visualizza le righe iniziali di un file di testo. Vedi anche `cat`, `less`, `more`, `tail`.

`hostname` Visualizza o imposta il nome (*hostname*) della macchina Unix.

`jobs` Visualizza l’elenco dei processi attivi eseguiti nella *shell* corrente. Vedi anche `ps`.

`kill` Provoca l’interruzione dell’esecuzione del processo specificato mediante il suo *Process ID* o mediante il *job number* (preceduto dal simbolo “`%`”). Ad esempio: “`kill %2`”.

`ksh` Korn Shell. È un interprete che esegue i comandi letti da *standard input* o da un file. Vedi anche `bash`, `csh`, `sh`, `tcsh`.

`last` Visualizza gli ultimi collegamenti effettuati dagli utenti sul sistema.

`latex` Sistema di composizione tipografica di documenti. Converte un file di testo scritto in formato LaTeX in un file DVI (device independent). Ad esempio: “`latex tesi.tex`”. Vedi anche `tex`, `xdvi`.

`less` Analogo a `more`. Vedi anche `cat`, `head`, `tail`, `view`.

`logout` Termina l’esecuzione della *shell* di login e conclude la sessione di lavoro dell’utente, scollegandolo dal sistema.

`lpq` Visualizza la coda di stampa; ad ogni elemento della coda di stampa è associato un *job number* ed un proprietario.

`lpr` Invia il file specificato alla coda di stampa; tramite il *pipe* può anche essere utilizzato per mandare in stampa l’output di un altro programma. Ad esempio: “`lpr pippo.txt`” oppure “`cat pippo.txt | lpr`”.

`lprm` Rimuove dalla coda di stampa un file ancora non stampato; si deve specificare come parametro il *job number* del file da eliminare, ottenuto con il comando `lpq`. Ad esempio: “`lprm 127`”.

`ls` Visualizza i file contenuti nella directory specificata.

`lynx` Programma ipertestuale per navigare sul web (*web browser*); non gestisce la visualizzazione di immagini grafiche, quindi può anche essere utilizzato su un terminale alfanumerico. Vedi anche `mozilla`, `netscape`, `wget`.

`mail` Programma per la lettura e la spedizione di messaggi di posta elettronica. Vedi anche `elm`, `pine`.

`man` Visualizza la pagina di manuale del comando specificato; ad esempio: “`man mail`”. Vedi anche `xman`.

`mkdir` Crea la directory specificata. Ad esempio: “`mkdir tesi`”. Vedi anche `rmdir`.

`more` — Visualizza il file specificato, introducendo una pausa alla fine di ogni schermata; mediante il *pipe* può anche essere usato come filtro dell'output di un altro programma. Ad esempio: "`more pippo.txt`" oppure "`cat pippo.txt | more`". Vedi anche `cat`, `head`, `less`, `tail`, `view`.

`mozilla` — Web *browser* grafico per l'ambiente X Window. Vedi anche `lynx`, `netscape`, `wget`.

`mv` — Sposta o rinomina il file specificato. Il formato del comando è "`mv` [*opzioni*] *origine destinazione*", dove *origine* è il nome del (dei) file da spostare, mentre *destinazione* è il nome del file o della directory in cui spostare il (i) file *origine*. Ad esempio: "`mv pippo.* ./src`".

`netscape` — Web *browser* grafico per l'ambiente X Window. Vedi anche `lynx`, `mozilla`, `wget`.

`nslookup` — Interroga un DNS server per risolvere gli indirizzi IP.

`oclock` — Orologio per X Window. Vedi anche `clock`, `date`, `xclock`.

`passwd` — Consente di cambiare la *password* di login. In ambiente di accounting centralizzato NIS (*Network Information Service – Yellow Pages*) si utilizza il comando `yppasswd`.

`pico` — Semplice editor di file di testo. Vedi anche `emacs`, `vi`.

`pine` — *Program for Internet News and Email.* Programma per la lettura e la spedizione di posta elettronica. Vedi anche `elm`, `mail`.

`ping` — Verifica se un certo *host* è raggiungibile sulla rete dalla macchina locale.

`ps` — Visualizza l'elenco dei processi attivi. Vedi anche `jobs`.

`pwd` — Visualizza il nome della directory corrente.

`rm` — Cancella i file specificati. Ad esempio: "`rm ./src/*`".

`rmdir` — Rimuove la directory specificata, che deve essere vuota (non deve contenere file o subdirectory). Ad esempio: "`rmdir src`". Vedi anche `rm`.

`rtin` — Programma per la lettura delle news Usenet. Vedi anche `pine`, `tin`.

`scp` — Esegue la copia di file in modalità sicura (cifrata) tra la macchina locale ed un *host* remoto. Vedi anche `ftp`, `ssh`.

`screen` — Consente di aprire più finestre virtuali su uno stesso terminale alfanumerico.

`set` — Sotto C Shell imposta il valore di una variabile. Vedi anche `setenv`.

`setenv` — Sotto C Shell imposta il valore di una variabile di ambiente. Vedi anche `export`, `set`.

`sh` — Bourne Shell. È un interprete che esegue i comandi letti da *standard input* o da un file. Vedi anche `bash`, `csh`, `ksh`, `tcsh`.

`shutdown` — Termina l'esecuzione di tutti i processi e spegne il sistema.

`sort` Ordina alfabeticamente i dati contenuti in una lista contenuta in un file o letta dallo *standard input*; mediante il *pipe* può essere usato anche come filtro per l'output di un altro comando. Ad esempio: "`sort lista.txt > lista.ord.txt`".

`ssh` Attiva una sessione di lavoro in modalità sicura (cifrata) verso un server remoto. Vedi anche `scp`, `telnet`.

`ssh-keygen` Genera la coppia di chiavi pubblica e privata per l'autenticazione con chiavi di cifratura asimmetriche in una sessione ssh/ssh2.

`tail` Visualizza le righe finali di un file di testo. Vedi anche `cat`, `head`, `less`, `more`.

`talk` Permette di comunicare in modo interattivo con un altro utente. Il formato del comando è il seguente: "`talk` *username* [*ttynn*]", dove *username* è il nome dell'utente ed il parametro opzionale *ttynn* è il nome del terminale su cui è collegato l'utente con cui si desidera comunicare. Ad esempio: "`talk liverani tty2`".

`tar` Tape Archive. Consente di archiviare in un unico file (che di default viene memorizzato su nastro) più file e directory. Con `tar` è anche possibile effettuare l'operazione inversa, cioè estrarre da un file di archivio (in genere con estensione "`.tar`") o da un nastro i file originali. Ad esempio: "`tar xfv pippo.tar`".

`tcsh` C Shell estesa. È un interprete che esegue i comandi letti da *standard input* o da un file. Vedi anche `bash`, `csh`, `ksh`, `sh`.

`telnet` È un programma con cui è possibile sfruttare il protocollo Telnet per utilizzare un sistema remoto collegato in rete (ad esempio su Internet). Vedi anche `ssh`.

`tex` È il comando per eseguire il programma di composizione tipografica di documenti TeX. Converte un file di testo in formato TeX in un file in formato DVI (device independent). Ad esempio: "`tex tesi.tex`". Vedi anche `latex`, `xdvi`.

`tin` Programma per la lettura delle news Usenet. Vedi anche `pine`, `rtin`.

`traceroute` Visualizza il percorso di rete tra la macchina locale ed un *host* remoto.

`tty` Visualizza il nome del terminale con cui l'utente è collegato al sistema.

`uncompress` Decomprime un file compresso con `compress` (vedi).

`unzip` Decomprime un file in formato zip; ad esempio: "`unzip pippo.zip`"; con il comando "`unzip -v pippo.zip`" si visualizza l'elenco dei file archiviati nel file `pippo.zip` in formato compresso. Vedi anche `zip`.

`vi` Editor di file di testo. Vedi anche `emacs`, `pico`.

`view` Esegue `vi` in modalità di sola lettura. Vedi anche `less`, `more`.

`w` Visualizza l'elenco degli utenti collegati al sistema. Vedi anche `finger`, `who`.

`wall` Invia un messaggio sul terminale di tutti gli utenti collegati al sistema. Vedi anche `write`.

`wc` Conta il numero di caratteri, parole o righe contenute in un file di testo.

`wget` Preleva un file da un server web utilizzando il protocollo HTTP. Vedi anche `lynx`, `mozilla`, `netscape`.

`who` Visualizza l'elenco degli utenti collegati al sistema. Vedi anche `finger`, `w`.

`whoami` Visualizza lo *username* dell'utente.

`whois` Interroga il database WHOIS per reperire informazioni relative all'organizzazione che ha registrato un certo dominio Internet.

`write` Consente di comunicare con altri utenti collegati al sistema, visualizzando sullo schermo del loro terminale il messaggio digitato. Ad esempio: "`write marco ttyp1`". Vedi anche `wall`.

`xcalc` Calcolatrice in ambiente X Window. vedi anche `bc`.

`xclock` Orologio classico per X Window. Vedi anche `clock`, `oclock`.

`xdvi` Programma in ambiente X Window per visualizzare documenti in formato DVI. Ad esempio: "`xdvi tesi.dvi &`". Vedi anche `dvips`, `latex`, `tex`.

`xedit` Semplice editor di testo in ambiente X Window.

`xeyes` Gli occhi che seguono il mouse sotto X Window.

`xhost` Imposta i permessi di accesso al server grafico X Window.

`xlock` Screen saver con possibilità di bloccare il terminale sotto X Window; ad esempio "`xlock -nolock -mode pyro`". Vedi anche `xset`.

`xlogo` Visualizza il logo di X Window in una finestra grafica.

`xman` Permette di sfogliare le pagine di manuale sotto X Window mediante una comoda interfaccia grafica con menù selezionabili con il mouse. Vedi anche `man`.

`xpdf` Visualizza e stampa documenti in formato PDF sotto X Window. Vedi anche `acroread`.

`xrdb` Modifica la configurazione delle applicazioni grafiche in ambiente X Window.

`xset` Imposta lo *screen saver* per la sessione di lavoro in ambiente X Window. Vedi anche `xlock`.

`xsetroot` Imposta alcuni parametri di configurazione dell'ambiente X Window; ad esempio con il comando "`xsetroot -solid cadetblue`" si imposta il colore dello sfondo.

`xterm` Emulazione di terminale alfanumerico sotto X Window.

`xv` Programma per la visualizzazione e l'elaborazione di immagini grafiche in ambiente X Window. Vedi anche `display`.

`yes` Visualizza indefinitamente il carattere "`y`".

`zip` Comprime file in formato zip. Permette di archiviare in formato compresso più file e directory in un unico file compattato. La sintassi del comando è la seguente: "`zip` [*opzioni*] *file.zip* file1, file2, ...", dove *file.zip* è il nome del file compresso di destinazione e *file1*, *file2*, ... sono i file da archiviare in *file.zip*. Ad esempio: "`zip src.zip ./src/*`". Vedi anche `compress`, `gzip`, `unzip`.

Appendice B

Elenco alfabetico delle sigle

ANSI	American National Standard for Information Systems
ASCII	American Standard Code for Information Interchange
BMP	Bitmap
BSD	Berkeley System Distribution
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
DEC	Digital Equipment Corporation
DMZ	Area non protetta di una rete (demilitarizzata)
DOS	Disk Operating System
DVI	Device Independet
EFF	Electronic Fronteer Foundation
FSF	Free Software Foundation
FTP	File Transfer Protocol
FVWM	F(?) Virtual Window Manager
GID	Group ID
GIF	Graphic Interchange Format
GNU	Gnu's Not Unix
GUI	Graphical User Interface
HP	Hewlett Packard
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IBM	International Business Machines
IMAP	Internet Message Access Protocol
JPEG	Join Photographic Expert Group

MS-DOS	Microsoft Disk Operating System
MWM	Motif Window Manager
NIS	Network Information Service (anche “Yellow Pages”)
NNTP	Network News Transport Protocol
NTP	Network Time Protocol
OLVWM	Open Look Virtual Window Manager
OLWM	Open Look Window Manager
PARC	Palo Alto Research Center
PC	Personal Computer
PDF	Portable Document Format
PID	Process ID
POP3	Post Office Protocol versione 3
RISC	Reduced Istruction Set Computer
SGI	Silicon Graphics Inc.
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
SSH	Secure Shell
SSH2	Secure Shell versione 2
SUN	Sun Microsystems (Stanford University Network)
SVR4	System V Release 4
TCP/IP	Transfer Control Protocol - Internet Protocol
TWM	Tab Window Manager
URL	Universal Resource Locator
WWW	World Wide Web

Bibliografia

[1] G. Anderson, P. Anderson, *The Unix C shell field guide*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986

[2] S. R. Bourne, *Unix System V*, Addison-Wesley, Milano, 1990

[3] B. Fox, *Bash Features*, Free Software Foundation, 1991, (reperibile su Internet[1])

[4] K. Hunt, *TCP/IP Network Administration*, O'Reilly, Sebastopol, CA, 1992

[5] B. W. Kernighan, R. Pike, *The UNIX programming environment*, Prentice-Hall, Englewood Cliffs, New Jersey, 1984

[6] B. W. Kernighan, D. M. Ritchie, *Linguaggio C*, Pearson – Prentice Hall, Milano, 1988

[7] D. E. Knuth, *The TEXbook*, Addison-Wesley Publishing Company, Reading, Mass., 1986

[8] L. Lamport, *LATEX A Document Preparation System*, Addison-Wesley Publishing Company, Reading, Mass., 1986

[9] R. Morgan, H. McGilton, *Il sistema operativo Unix System V*, Mc Graw-Hill, Milano, 1988

[10] P. Norton, H. Hahn, *Unix*, Mondadori Informatica, Vicenza, 1992

[11] M. J. Rochkind, *Advanced Unix programming*, Prentice-Hall, Englewood Cliffs, New Jersey, 1985

[12] J. J. Valley, *La grande guida Unix*, Jackson Libri, Milano, 1993

[13] M. Welsh ed altri, *Linux Installation and Getting Started*, The Linux Documentation Project, Agosto 1998 (reperibile su Internet)

[14] L. Wirzenius ed altri, *The Linux System Administrator's Guide*, The Linux Documentation Project, 2003 (reperibile su Internet)

[1] Si intende dire che una copia tel testo in formato PostScript o ASCII può essere reperita presso numerosi siti Internet mediante FTP anonimo.

Indice analitico